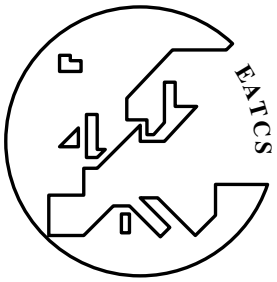
ISSN 0252–9742

Bulletin

of the

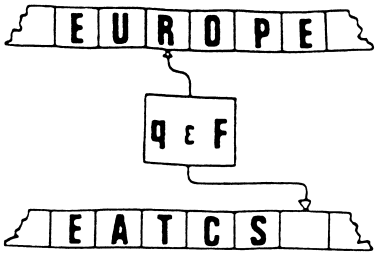
European Association for Theoretical Computer Science

EATCS



Number 148 February 2026

Council of the European Association for Theoretical Computer Science



President:	Giuseppe F. Italiano	Italy
Vice Presidents:	Antoine Amarilli	France
	Inge Li Gørtz	Denmark
Treasurer:	Gabriele Fici	Italy
Bulletin Editor:	Stefan Schmid	Germany

Antoine Amarilli	France	Giuseppe F. Italiano	Italy
Alkida Balliu	Italy	Kasper Green Larsen	Denmark
Ivona Bezakova	USA	Yannic Maus	Austria
Thomas Colcombet	France	Anca Muscholl	France
Anne Driemel	Germany	Yasamin Nazari	Netherlands
Funda Ergün	USA	Charles Paperman	France
Javier Esparza	Germany	Eva Rotenberg	Denmark
Gabriele Fici	Italy	Jukka Suomela	Finland
Pawel Gawrychowski	Poland	Szymon Torunczyk	Poland
Inge Li Goertz	Denmark	Bianca Truthe	Germany

Past Presidents:

Maurice Nivat	(1972–1977)	Mike Paterson	(1977–1979)
Arto Salomaa	(1979–1985)	Grzegorz Rozenberg	(1985–1994)
Wilfred Brauer	(1994–1997)	Josep Díaz	(1997–2002)
Mogens Nielsen	(2002–2006)	Giorgio Ausiello	(2006–2009)
Burkhard Monien	(2009–2012)	Luca Aceto	(2012–2016)
Paul Spirakis	(2016–2020)	Artur Czumaj	(2020–2024)

Secretary Office:	Dmitry Chistikov	UK
	Efi Chita	Greece

EATCS Council Members

email addresses

Antoine Amarilli . . . . . a3nm@a3nm.net
Alkida Balliu . . . . . alkida.balliu@gssi.it
Ivona Bezakova . . . . . ib@cs.rit.edu
Thomas Colcombet . . . . . Thomas.Colcombet@irif.fr
Anne Driemel . . . . . driemel@cs.uni-bonn.de
Funda Ergün . . . . . chair.sigact@sigact.acm.org
Javier Esparza . . . . . esparza@in.tum.de
Gabriele Fici . . . . . gabriele.fici@unipa.it
Pawel Gawrychowski . . . . . awry@cs.uni.wroc.pl
Inge Li Goertz . . . . . inge@dtu.dk
Giuseppe F. Italiano . . . . . giuseppe.italiano@uniroma2.it
Kasper Green Larsen . . . . . larsen@cs.au.dk
Yannic Maus . . . . . yannic.maus@tugraz.at
Anca Muscholl . . . . . anca@labri.fr
Yasamin Nazari . . . . . y.nazari@vu.nl
Charles Paperman . . . . . charles.paperman@univ-lille.fr
Eva Rotenberg . . . . . eva@rotenberg.dk
Stefan Schmid . . . . . stefan.schmid@tu-berlin.de
Jukka Suomela . . . . . jukka.suomela@aalto.fi
Szymon Torunczyk . . . . . szymtor@mimuw.edu.pl
Bianca Truthe . . . . . bianca.truthe@informatik.uni-giessen.de

Bulletin Editor: Stefan Schmid, Berlin, Germany
Cartoons: DADARA, Amsterdam, The Netherlands

The bulletin is entirely typeset by pdfTEX and ConTEXt in TXfonts.

All contributions are to be sent electronically to

bulletin@eatcs.org

and must be prepared in LaTeX 2ε using the class `beatcs.cls` (a version of the standard LaTeX 2ε `article` class). All sources, including figures, and a reference PDF version must be bundled in a ZIP file.

Pictures are accepted in EPS, JPG, PNG, TIFF, MOV or, preferably, in PDF. Photographic reports from conferences must be arranged in ZIP files layed out according to the format described at the Bulletin's web site. Please, consult `http://www.eatcs.org/bulletin/howToSubmit.html`.

We regret we are unfortunately not able to accept submissions in other formats, or indeed submission not *strictly* adhering to the page and font layout set out in `beatcs.cls`. We shall also not be able to include contributions not typeset at camera-ready quality.

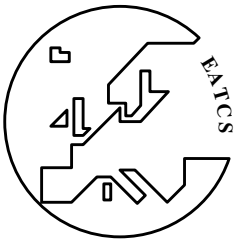
The details can be found at `http://www.eatcs.org/bulletin`, including class files, their documentation, and guidelines to deal with things such as pictures and overfull boxes. When in doubt, email `bulletin@eatcs.org`.

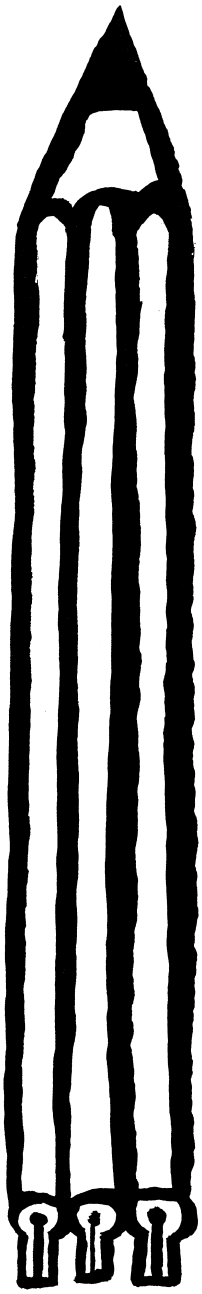
Deadlines for submissions of reports are January, May and September 15th, respectively for the February, June and October issues. Editorial decisions about submitted technical contributions will normally be made in 6/8 weeks. Accepted papers will appear in print as soon as possible thereafter.

The Editor welcomes proposals for surveys, tutorials, and thematic issues of the Bulletin dedicated to currently hot topics, as well as suggestions for new regular sections.

Table of Contents

EATCS Matters





Dear Colleagues,

As we begin 2026, I would like to send my warmest wishes to all of you. May this year bring good health, stimulating ideas, bold research directions, and many opportunities to meet, exchange, and collaborate. Our field thrives on intellectual curiosity and collective energy, and I look forward to continuing this journey together, promoting theoretical computer science in its broadest sense and strengthening the role of EATCS within our community.

The new year also brings important opportunities to recognize excellence. Calls for nominations are currently open for several EATCS-sponsored awards: the EATCS Award, the Presburger Award, the EATCS Distinguished Dissertation Award, and the EATCS Fellows. In addition, EATCS proudly co-sponsors major international prizes such as the Gödel Prize, the Alonzo Church Award, and the Dijkstra Prize. I warmly encourage you to take a moment to nominate outstanding colleagues. Recognition is not only a celebration of individual achievement; it is also a way of telling the story of our field and highlighting the ideas that shape its future.

One of the moments I am particularly looking forward to this year is the 53rd International Colloquium on Automata, Languages, and Programming (ICALP 2026), which will take place at Royal Holloway, University of London from July 7-10. As you may have seen, CORE has recently upgraded ICALP's ranking from A to A.*

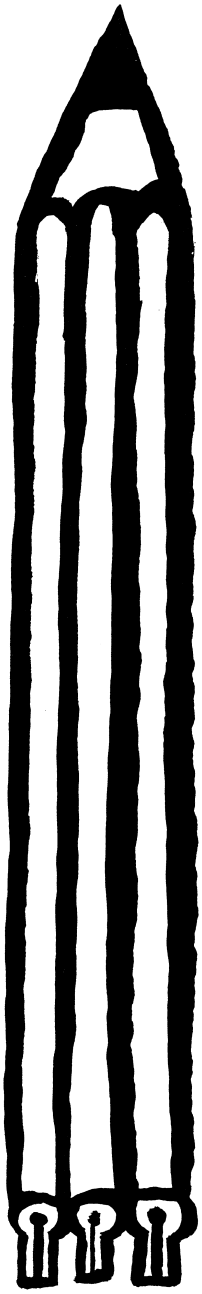
This recognition reflects the sustained effort of many colleagues over the years to preserve and strengthen the scientific excellence of ICALP, consolidating its role as the flagship conference of EATCS. This year, ICALP will be co-located with the ACM Symposium on Principles of Distributed Computing (PODC) and the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), a combination that promises an intellectually vibrant and truly memorable scientific gathering. As every year, ICALP will provide a unique forum where ideas meet, collaborations are born, and communities intersect.

The year will also feature several other outstanding EATCS-affiliated events:

** MFCS 2026: International Symposium on Mathematical Foundations of Computer Science - Paris, France, August 24-28*
** ESA 2026: European Symposium on Algorithms - L'Aquila, Italy, August 31-September 4*
** DISC 2026: International Symposium on Distributed Computing - Rome, Italy, November 9-13*

I hope to meet many of you at these conferences and workshops. These occasions are far more than entries on a calendar - they are where our community renews itself, exchanges ideas, and reflects on how we can further strengthen the impact and visibility of theoretical computer science.

Over the past year, the EATCS General Assembly has taken important steps on several fronts. There is certainly more that we could accomplish with additional resources, but I am deeply grateful to all



our members for their continued support. Thanks to you, we can publish the Bulletin as a freely accessible resource, support prestigious awards, and sustain a range of initiatives that serve the TCS community.

Our guiding principle is simple: to give back to the community what the community gives to us through its trust and membership. If you are not yet a member of EATCS, I warmly invite you to join - and to encourage your colleagues and students to do the same.

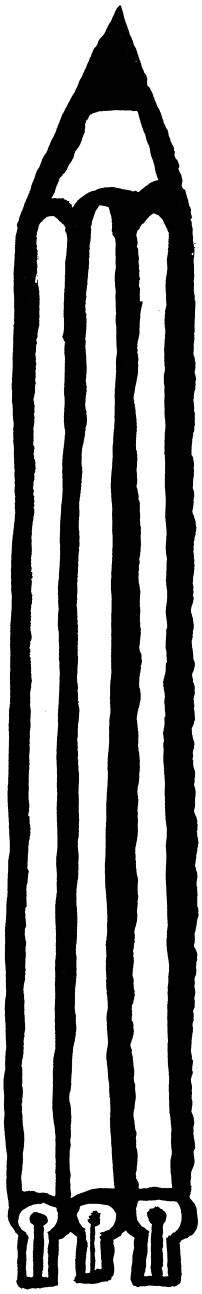
Finally, I would very much like to hear from you. EATCS exists to serve the theoretical computer science community, and your ideas, suggestions, and even criticisms are invaluable. Please do not hesitate to reach out. Together, we can continue pushing the boundaries of our discipline and ensure that its voice remains strong and influential.

Wishing you a wonderful and inspiring 2026. I very much hope to see you soon.

With best regards,

Giuseppe F. Italiano
Luiss University, Rome
President of EATCS
president@eatcs.org

February 2026



Dear EATCS member!

What do rotating drum memories, SAT solvers, and Broadway scores played in alphabetical order have in common? In this issue's Interview Column, Don Knuth joins us to share his experiences and advice with the Bulletin's readers.

In the Viewpoint Column, Wojciech Czerwiński offers a personal report on a discussion panel at Highlights on the future of research in the automata, games, and logic community. I am also delighted to welcome our new Viewpoint Column editor, Anca Muscholl. Thank you, Anca - it is wonderful to have you on board!

In the TCS on the Web Column, Nutan Limaye introduces her podcast Life of a Researcher, available on Spotify and Podcast Addict, featuring conversations with researchers across fields and career stages.

This issue also launches a new feature: the Conference Reports Column, edited by Laurent Feuilloley. Welcome aboard, Laurent, and thank you for joining the editorial team! In his inaugural column, Laurent reports on DISC 2025 in Berlin and invites contributions - please get in touch with him if you would like to contribute.

In the Machine Learning Column, Pablo Barceló and David Saulpic interview Santosh Vempala, exploring explanations of hallucinations from a theoretical computer science perspective.

In the Computational Complexity Column, Edward Pyne and Roei Tell survey recent results on applying hardness-vs.-randomness

techniques to the design of low-space algorithms.

In the Logic in Computer Science Column, Gyeongwon Jeong, Seonghun Park, and Hongseok Yang introduce Razborov's flag algebras and explain their applications in extremal graph theory.

The Concurrency Column surveys the theoretical foundations of runtime monitoring, covering classic regular properties, data-dependent properties, and hyperproperties. In the Education Column, Ryan E. Dougherty and Tim Randolph provide an overview of theory of computing education in the United States.

The Bulletin further includes a report on Yurifest 2025, an obituary for Gilles Dowek, and a report on CIAA 2025.

Last but not least, a friendly reminder about our new YouTube channel! Please contact Sophie and Ian if you are interested in contributing a video.

Enjoy the new issue of the Bulletin!

Stefan Schmid, Berlin
February 2026

Institutional Sponsors

CTI, Computer Technology Institute & Press "Diophantus"
Patras, Greece

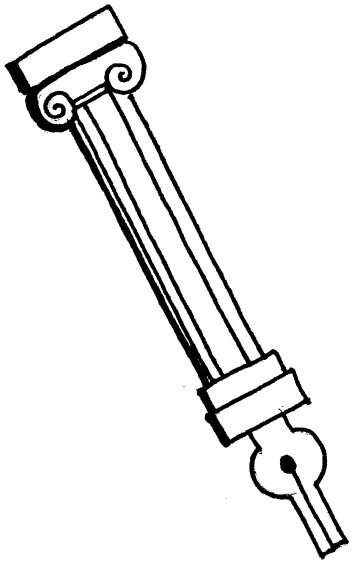
CWI, Centum Wiskunde & Informatica
Amsterdam, The Netherlands

MADALGO, Center for Massive Data Algorithmics
Aarhus, Denmark

Microsoft Research Cambridge
Cambridge, United Kingdom

Springer-Verlag
Heidelberg, Germany

EATCS
Columns



1

The Interview Column

BY

Chen Avin and Stefan Schmid

Ben Gurion University, Israel and TU Berlin, Germany
{chenavin,schmiste}@gmail.com

KNOW THE PERSON BEHIND THE PAPERS

Today: Don Knuth

Bio: *Don Knuth is Professor Emeritus at Stanford University and the author of the landmark series* The Art of Computer Programming. *His contributions to algorithms, analysis, formal languages, TeX, literate programming, and combinatorics have shaped modern computing. He is a Turing Award laureate and continues to be an active researcher, writer, and musician.*

Figure 1: Photo credits: Rajan P. Parrikar

Stefan Schmid: Is there a photo you like to share with us, for the interview column?

Don Knuth: You can just pick something from my homepage, there's actually a whole collection of "downloadable graphics" with more or less unusual photos of me. Maybe you can take a picture that shows me in a natural setting, perhaps near an organ or a piano, which are important parts of my life.

Stefan Schmid: Can you please tell us something about you that most of the readers of your papers probably don't know?

Don Knuth: One thing is my lifelong connection to music. I started playing the organ when I was 12 years old — my father had a piano at home because he was teaching students. That led to a deep love of organ music. I eventually wrote a large organ composition, which was performed on my 80th birthday. It mixes about 18 different musical styles; I knew it would probably be the only big piece I ever wrote, so I decided not to hold back. These days I'm working my way, on the piano, through Broadway scores in alphabetical order. Recently I played through *Carousel*, then *Damn Yankees*, and most recently *Fiddler on the Roof*. I also helped pay for the organ at the church I attend, using money from the Kyoto Prize, and I have an organ at home. Music is still a big part of my daily life.

Stefan Schmid: Can you tell us the story behind one of your papers? For example one that took very long or one which evolved in an unexpected way?

Don Knuth: A nice example goes back to my first paper in the *Journal of the ACM*. It was about how to place instructions on a rotating drum so as to minimize latency. Depending on where instructions were placed, you could lose on the order of 50 milliseconds if they were in the "wrong" place. I set up a small example as an integer programming problem, and I think I was perhaps the first person to implement Ralph Gomory's algorithm for integer programming — he told me so, at least. Unfortunately, our computer at the time was extremely small and slow, so the problem was hard to solve. It was finally cracked only decades later, and I recently saw a paper about "Knuth's old latency problem," now solved very quickly by modern IP solvers. So that's a paper whose story stretches from drum memories in the early 1960s all the way to 21st-century optimization software.

Another kind of "paper that got out of hand" is not a paper but the sections in *The Art of Computer Programming* on BDDs, SAT solvers, and exact cover with colors (XCC). None of these topics existed when I drafted the table of contents in the 1960s, but they turned out to be so important that, for example, the SAT-solver section alone ended up with almost 500 exercises.

Stefan Schmid: When (or where) is your most productive working time (or place)?

Don Knuth: Historically, my most productive period was probably around 1966, when I was at Caltech and working both on algorithms and on Volume 2 of *The Art of Computer Programming*. I would wake up in the morning saying to myself, "Here's what I'm going to finish today," and then work until it was finished, often late at night. I had two young children at the time, and I did a lot of work at home with the television on in the background. It was highly productive in terms of "algorithms per hour," but it was also completely unsustainable: I ended up in the

hospital. You can actually locate exactly where I was in Volume 2 at that moment — there is a sneaky index entry under "brute force" pointing to a Latin phrase on the page I was working on.

Nowadays I still work mostly at home. I usually swim in the morning and, while swimming, think about what I'm going to write later. Then I come back and spend the rest of the day writing, programming, or debugging. Wednesdays are partly reserved for gardening and laundry.

Stefan Schmid: How do you choose what to work on? And what kind of impact are you hoping your work will have? Did this change over the course of your career?

Don Knuth: In 1962 I wrote down a tentative table of contents for what became *The Art of Computer Programming*. That list, in a sense, has defined my research agenda ever since. I simply move through it steadily: last week I might have been on one page, this week on the next. Of course, the contents have evolved — new topics like BDDs, SAT solvers, and parameterized complexity have reshaped some chapters — but the overall roadmap is still there.

As for impact, when I write about a topic I try to ask myself, "Will this still be interesting 50 years from now?" I'm not very good at predicting the future, but I do my best to choose subjects that will remain valuable in the long run. My goal has always been to provide a deep, reliable foundation that future generations can build on.

Stefan Schmid: What do you do when you get stuck with a research problem? How do you deal with failures?

Don Knuth: Sometimes I write to colleagues or talk to people at lunch at Stanford, explaining exactly where I'm stuck. In other cases I simply put the problem aside and hope that, months or years later, I'll return with a fresh perspective. Only rarely — maybe two or three times in my whole life — has a solution come to me in the middle of the night. More often, I write a program to get more facts about the problem, and the very process of programming helps clarify my thoughts in some magical way. Of course failures greatly outnumber successes.

A concrete example: I recently spent four hours trying to understand an early survey paper on what we now call treewidth. The quantifiers ("for some," "for every") were used in such an ambiguous way that I simply couldn't tell what was really meant. In the end I pencilled a note on the paper saying, "I can't make head nor tail of this," dated it, and put it away — with a few clues I had extracted in case I ever return. Then I went back to working on Hamiltonian cycles, where I could actually make progress.

Stefan Schmid: Do you have any advice for young researchers? In what should they invest time, what should they avoid?

Don Knuth: My main advice is: Try to work on things that are genuinely interesting and somewhat unique to you, rather than on topics that are "hot" because the community says so. Everyone has a different background and set of experiences, which means each person is uniquely qualified to solve certain problems. You usually do not know in advance which problems those are, so it's important to get to know your own strengths and weaknesses.

I also wish there were more emphasis on algorithms that actually compute things in practice, not just in asymptotic theory. There are many papers shaving off a factor like the cube root of $\log\log n$ when $\log\log n$ is never more than five. Such results can be interesting because they expose bottlenecks, but too often we stop after proving polynomial vs exponential time and forget that constants matter enormously.

So: Invest time in problems where you can really understand the phenomena deeply, where you can implement and test algorithms, and where your work might remain meaningful decades from now. Prefer a problem that "has your name on it" because of your personal background. Avoid chasing fashions just because they're fashionable.

Stefan Schmid: What research topics or approaches do you wish the TCS community in general would study more?

Don Knuth: I'd like to see more work on what can be *really* computed, not just in principle. Our field has many beautiful complexity results, but it's easy to stop at labels like "polynomial time" or "exponential time" and forget that the constant factors and actual implementability are crucial.

Of course I don't restrict attention entirely to questions of efficient real-world computation. Sometime a purely theoretical question that has a negligible impact on practical programs is of great interest, often because of its historical value or the way it helps us to focus on general techniques that are widely useful also in other contexts. For example, the study of optimum addition chains has always fascinated me: It is one of the earliest fundamental computational problems to have been investigated, and its theory leads to numerous insights — even though I'll never actually want to compute $x^{5784689}$ with the fewest possible multiplications.

I've recently spent a lot of time on exact cover, SAT, and now exact cover with colors (XCC), which have turned out to be tremendously powerful tools for solving concrete combinatorial problems such as graph embedding.

At the same time, I admire developments in parameterized complexity and kernelization, which have enriched our understanding of hard problems like vertex cover. My wish is not to replace one area with another, but to keep an eye on the interplay between elegant theory and robust, implementable algorithms.

Stefan Schmid: How do you approach evaluating research papers? What about

grant proposals?

Don Knuth: I'm usually interested mostly in learning the techniques that led to a solution to a specific problem, much more than in the actual solution itself; those techniques will help me with future problems. Before I turn each page, I try to guess what will be on that page. I reformulate the problem in other notations, and I try to solve it myself before reading on to see what the author has actually done. Of course I usually fail; but this exercise has prepared me to learn from my failures, and to appreciate what the author has achieved.

I am especially skeptical of proofs that consist of hundreds or thousands of unverified cases in an appendix or technical report. In one area of graph classes, for example, I found typographical errors in the cases I checked and no code to back up the claims, which made the results unconvincing to me.

So I value papers where the authors either provide implementations or at least make it reasonably clear how their algorithms would be programmed and tested. The same general criteria would apply if I were evaluating grant proposals.

Stefan Schmid: Do you see a main challenge or opportunity for theoretical computer scientists for the near future?

Don Knuth: One continuing challenge is to maintain high standards of rigor while engaging seriously with real-world computational problems. We have powerful tools — SAT solvers, IP solvers, XCC solvers, parameterized algorithms, and many others — and there are huge opportunities in understanding when each tool is appropriate and how they complement one another.

But in general, I'm not good at predicting the future; in fact, there are many developments in computer science that surprised me, and hardly any I foresaw. My own focus remains on topics where I feel I can contribute durable insights.

Stefan Schmid: Is there a blog, podcast, book, or movie you like particularly?

Don Knuth: I haven't had a television set for about 40 years, and I don't follow blogs or podcasts; I honestly don't know where people find the time. Reading the newspaper already takes more time than it should and tends to depress me.

However, my homepages include a page in the FAQ section where I list about forty books that I've especially enjoyed — mostly not about mathematics. One mathematical book I particularly like is Peter Winkler's *Mathematical Puzzles*. A new edition came out recently, and it's brilliant.

Stefan Schmid: Is there a nice anecdote from your career you would like to share with our readers?

Don Knuth: One favorite anecdote involves knight's tours on the chessboard. When I first learned about computers in the 1950s, one of the earliest problems I tried to tackle was: "How many knight's tours are there on an 8×8 board?" I

wrote a little program and quickly realized the search space was enormous, so I gave up.

Then, in the 1990s, Ingo Wegener and a coauthor used clever methods and modern computers to compute what they believed was the exact number, around 33 billion. When I saw their paper, I immediately noticed that their number was not divisible by 4, and it's not hard to prove that the count must be a multiple of 4. So I knew something was wrong.

I contacted my colleagues, and they eventually computed the correct number and found the error in the earlier calculation. The method itself was basically sound; there had just been a slip. For me this story nicely illustrates several themes: the power of computers and algorithms, the importance of simple invariants (like divisibility by 4), and the way mathematics, computation, and careful checking all interact.

Please complete the following sentences?

- *My first research discovery*
  was an "imaginary" number system based on the square root of -2, which I developed in high school and later published after submitting it to the Westinghouse Science Talent Search.
- *The key to being a happy academic*
  is to pretend that politics doesn't exist.

Stefan Schmid: Don, many thanks for this wonderful conversation. Any final words before we let you go?

Don Knuth: Thank you! I really enjoyed the questions you asked. And now I think I'll go fold the laundry — it's Wednesday, after all.

TCS ON THE WEB

BY

STEFAN NEUMANN

TU Wien
Erzherzog-Johann-Platz 1
1040 Vienna, Austria
stefan.neumann@tuwien.ac.at
https://neumannstefan.com

Nutan Limaye[1] is a Full Professor in Theoretical Computer Science at the IT University of Copenhagen. Her research is in computational complexity theory, with a particular focus on algebraic complexity and circuit lower bounds. She has received several awards, including a FOCS best paper award, and she was a keynote speaker at the Computational Complexity Conference 2024.

In this interview, we speak with Nutan about her podcast *Life of a Researcher* which is available on Spotify[2] and on Podcast Addict[3]. The podcast features long-form conversations with researchers across fields and career stages, with a particular focus on the lived reality of research: how people enter the profession, what sustains them, and how diverse backgrounds shape research trajectories. Our conversation explores the motivation behind the podcast, recurring themes that emerge across episodes, how guests are selected, and how topics such as neurodiversity fit naturally into the project.

[1] https://www.itu.dk/~nuli/
[2] https://open.spotify.com/show/4T8mDPAJdqJw1349X5l8Mt
[3] https://podcastaddict.com/podcast/life-of-a-researcher/4111736.

On the Life of a Researcher

A Conversation with Nutan Limaye

Nutan, thank you for agreeing to do this interview. I enjoy your podcast a lot. To start, could you give our readers a short overview of the podcast?

The podcast is called *Life of a Researcher*. The idea is to talk to researchers and learn why they are in research, how they got there, and what their trajectory was. For example: how they grew up, whether their early experiences influenced what they do now, and more generally how different people end up in research.

One motivation is that many of us grow up with a very stereotypical image of a researcher: someone working alone, not talking to anyone, having a sudden "eureka moment," and discovering something. That is certainly an image I had.

But when you actually do research, you realize it is not like this at all. People have very different backgrounds and perspectives. And in many areas, dynamic interaction is essential: collaboration, understanding other people's ideas, and communicating well. These aspects of research life are rarely part of popular communication about research. That is the viewpoint behind the podcast.

I like that you add dimensions people usually don't talk about, and I think that matters a lot for young researchers. If you had to give a pointer for early-career researchers: is there a particular episode you would recommend as a starting point?

It depends on who you are. Role models can be very important. If you feel alone—for example because you are a woman in a male-dominated field, or you are neurodivergent, or you are LGBTQ+—it can help to hear from people with related experiences.

I have been fortunate to host guests with a broad cross-section of backgrounds and experiences. So rather than naming one or two episodes, I would suggest looking at the short summaries in the episode list and choosing something that resonates with where you are right now.

Looking back at the episodes you have recorded so far: is there a theme or message that keeps coming up?

This is interesting because, by design, I try to capture variety, so the episodes are quite different.

But something that comes up again and again is that many guests have felt, at some point, "I'm not really fitting in." Even people who are now at the top of their research careers describe periods of confusion and self-doubt. And yet they come out on the other side.

This can be inspiring for young people, because they also doubt themselves. Of course, I don't want to paint everything as rosy: there is a selection bias, since I'm interviewing people who stayed in research and built careers. But perseverance comes up a lot. And also: they genuinely seem to enjoy what they do.

I ask guests at the end what they love about their work, and what they would remove from their research life if they could. For the second question, people often have to think hard—there is not an immediate long list. That tells you something.

That leads nicely to the next question. How do you find your guests? Do you have a process?

Since I started thinking about the podcast, I've been on the lookout. Some guests come through my network, of course. But sometimes it's serendipity.

For example, I invited Geraldine Fitzpatrick after attending a leadership course she taught. I was pleasantly surprised by the real rigor with which she offered the leadership course. And her background intrigued me: she works in human-computer interaction, and earlier she trained as a nurse. That combination, and her perspective as a research leader, made me want to have her on the podcast.

So I often look for people with unusual paths or distinctive perspectives, and then I reach out. Some interesting people decline because they don't feel comfortable speaking publicly in this format. That happens too.

One thing I appreciate is that you bring in perspectives beyond TCS. You bring in guests from HCI and beyond. How have these interdisciplinary conversations influenced the way you think about research, or about theoretical computer science as a field?

I'm not sure it has changed how I think about computer science itself, but it has made me more aware of how different the cultures are across fields.

When I interview someone outside TCS, I prepare more. I try to understand what their day-to-day looks like and what matters in their context—maybe grants are more important for them, or workshops, or other practices that differ from ours.

I also think it's useful for a mostly TCS audience to hear about challenges in other areas. It can reduce the "grass is greener" feeling and help us appreciate what works well in our community—while still being able to critique what should improve.

You also cover topics like neurodiversity, ADHD, and dyslexia. You seem to make a deliberate effort to include these experiences. What led you to that, and is there a key learning you would share?

First, it fits naturally with the podcast theme: these are life experiences that can shape what it means to be a researcher.

Second, I've been sensitized to these challenges through personal experience and through teaching. As a PhD student, I had a close friend with clinical depression. At the time, we tried to be supportive, but I don't think we truly understood how hard it was for her to complete her PhD. It is only much later that it really sinks in what that must have taken.

Now, as a professor and teacher, I meet students with a wide range of challenges. I cover basic accessibility practices, but there is a lot we don't know. Having open conversations—when guests are willing—helps build understanding. I'm not trying to push a large agenda; it's mainly about awareness, first for myself and then perhaps for listeners over time.

And in TCS specifically, I feel the research problems themselves can be conducive to different ways of thinking. People with ADHD or other forms of neurodivergence may bring distinctive creativity to combinatorial and abstract problems. Historically, there were many brilliant people with challenges that were unnamed at the time. So I also want the message to be: this is not a disqualifier. You can be an excellent researcher.

Has hosting the podcast changed how you approach your own research, teaching, or mentoring?

Not in a dramatic way—these things take time to internalize. But the conversations keep me open and attentive. And there are practical things I've learned: for example, some guests are excellent at compartmentalizing or multitasking, and their advice—often intended for early-career researchers—is useful for me too.

Is there anything else you want to share about the podcast?

I'm thinking about the next season and making some episodes more topical for early-career researchers—for example: how to write a paper, how to present at a conference or workshop, how to apply for grants, how to handle feedback, how to collaborate on drafts. These are practical deep-dives into aspects of research life.

This might become more TCS-focused simply because it is easier for me to find people in that community whose advice I can trust. But the goal would be resource creation—concrete help.

Before we conclude, could you tell us what your current research is about?

I'm a complexity theorist. Complexity theory studies limitations of resources: given a certain amount of time or space, what can't you compute within those bounds? In that sense it is a kind of dual view to algorithms.

In the last decade, I've worked mainly on algebraic complexity theory. There, you represent computational problems as polynomials and study the complexity of computing those polynomials: how many additions and multiplications are needed?

A core question is understanding the limits of computation for polynomials believed to be hard, analogous in spirit to P vs NP. In algebraic complexity, one famous problem is VP vs VNP. A central example is the permanent polynomial, which encodes counting perfect matchings in a graph. We know exponential upper bounds, but we don't know whether exponential size is necessary; it's widely believed to be.

One of my recent results shows that the permanent is provably hard for certain restricted circuit classes—for example, very shallow (constant-depth) circuits, even when they are highly parallel. The lower bound we get is super-polynomial, showing that polynomial size is not enough under those depth restrictions.

Thank you very much for the interview.

Conference Reports Column

by

Laurent Feuilloley

Laboratoire d'informatique en image et systèmes d'information (LIRIS)
CNRS and Université Lyon 1
43 Bd du 11 novembre 1918, 69622 Villeurbanne, France
laurent.feuilloley@cnrs.fr

BEATCS no 148

CONFERENCE REPORT ON DISC 2025

Laurent Feuilloley

Abstract

Dear readers, this is my first time as the editor of this conference reports column. I would like to thank Stefan for this opportunity. I'll write the first report myself, and it will be about DISC 2025. However, I will not be able to write all future reports alone — I will need your contributions!

As I worked on this report, I realized the challenge of this task. Should the focus be on the broader BEATCS readership, or specifically on the colleagues from my community who were unable to attend? Should you focus on the fun facts that we like to read, or on the science, that we should like to read? I'll try to do a bit of everything. If you are willing to write a report in the future, you will choose what you prefer.

What is DISC? DISC is a well-established conference whose topic is distributed computing, mostly of theoretical flavor. It is the flagship conference for this topic in Europe, and is sponsored by the EATCS.[1] It was established in 1985, shortly after its ACM counterpart, PODC. If you want to know more, read the story of DISC in the previous bulletin [1]!

The venue The conference took place in Harnack house in Berlin. It is actually quite far from central Berlin, but close to a large park, and remains easily accessible by public transport. You may find yourself visiting the venue in the future as Harnack House has previously hosted other TCS conferences, including SoCG 2022.

Harnack house was a major place for physics in early 20th century, and is sometimes referred to as the "German Oxford". Its grand lobby features portraits of well-known figures who once worked or visited there, including Lise Meitner, Max Planck, and Albert Einstein. It is a very fancy building, with a nice park though we couldn't enjoy it much because of the unfriendly weather..

[1] Conferences need not be EATCS-sponsored to be featured in this BEATCS column!

Figure 1: Harnack house, where DISC took place.

The science I had planned to write about quite a few talks, but I realized that I would either focus only on topics closely tied to my research or spend hours to decipher my notebook. So let me just say a few words about the invited talks.

Moni Naor gave a talk on his 1985 paper with Larry Stockmeyer, *What can be computed locally?* [2], which was awarded the Dijkstra award this year. The Dijkstra Prize is the most prestigious award in the PODC/DISC community. This paper introduced the notion of *locally checkable language*, a framework for problems whose solutions can be verified locally—such as graph coloring. This concept has been a very important for distributed graph algorithms in recent years. Moni shared an interesting backstory: the paper's origins trace back to a talk he attended on Ehrenfeucht-Fraïssé games — a tool from model theory for proving limits of first-order logic! Then the project went in a different direction, but it's nice to see that back in the days, track A and track B were so close. The awarded paper was co-authored by Larry Stockmeyer, who passed away in 2004. Stockmeyer had already been awarded a posthumous Dijkstra award in 2007, for his work on partial synchrony with Dwork and Lynch, and you might also know him for introducing the polynomial hierarchy.

François Le Gall gave an invited talk on quantum distributed algorithms. This is a theme that was quite strong at DISC, with a survey in a workshop, a paper in the main program and this invited talk. François told us about the centralized quantum algorithms, and then moved to distributed. One insight was that, while reducing communication through entanglement seems natural, the challenge was to have a purely local problem that would show quantum advantage even when the complexity measure is just the locality. This was achieved only very recently.

Ittai Abraham gave a talk called *Open Questions and Future Challenges in Fault Tolerant Distributed Computing*. The talk was both entertaining and thought-provoking, built around 10 claims — some bold, others more nuanced — explor-

ing the future of distributed computing, its research community, trust, AI, the role of formal proofs... I guess giving some bits of it here would not do justice to the talk, so I'd better refer you to the nice blog of Ittai for similar nice content: https://decentralizedthoughts.github.io/.

Business meeting and discussions Now, let's turn to the business meeting which was chaired by the steering committee chair, Hagit Attiya. I'll go into more detail here, because I like the conference design discussions, and also it is interesting to a broader audience.

The local organizers, Stefan Schmid and Joel Rybicki told us about what happened behind the scene and shared the usual statistics. here were significantly more participants than in previous years — 160, instead of 100-120. Maybe because Berlin is more central, easier to reach, or because the communication was better? It was not because registration was cheaper: people noted a clear increase in registration costs. Perhaps this is the trade-off for hosting the conference in a building where Meitner and Einstein have given talks.

Darek Kowalski, the PC chair, then gave us many insights about the selection process, and changes he had made.

The first change was that there was no PC meeting this year. (For those unfamiliar, a PC meeting typically occurs just before notifications, where the committee gathers for a 10-hour Zoom session to decide the fate of the remaining ~10 out of ~30 papers that are neither accepted nor rejected.) Instead, the process relied on extended online discussions via the submission platform, with papers grouped by topic for more focused discussions. I'm not a big fan of PC meetings, so I liked the idea. The downside, however, is that this approach required more time for discussions, pushing part of the PC work into the summer vacation period. This likely explains why Darek mentioned that recruiting strong PC members was more challenging this year.

The PC was sufficiently large that each member was responsible for only about 12 papers. Again, this looks good to me: I recently served on a PC with a much heavier load, and keeping track of all those papers exceeded my limited working memory!

A recurring topic at DISC/PODC is the rebuttal process: Should authors be given the chance to respond to reviews, even if it extends the timeline? Does it actually change something? This year, there was a rebuttal phase, and Darek said it changed the fate of four papers. Not a huge number, but not insignificant either. It also likely improved review quality, as there were fewer complaints about poorly written reviews."

Darek also decided to slightly increase the acceptance rate. Every year, there are strong papers that all relevant PC members appreciate but cannot accept due

to space constraints, so I see this as a positive change. The rationale behind this decision was that when the acceptance rate is low on year n, then there are less submission on year $n + 1$, which is unfortunate.

Finally, Darek chose to select a few papers that were labeled as *highlights*, in addition to the usual best papers, and that were presented in separate sessions. I heard positive and negative opinions about this idea. On the plus side, it is something nice to put on a CV, a way to draw researchers from topic A to attend a few talks about topic B, and anyway these papers would have been distinguished by being invited to the special issue. On the minus side, I heard "Ok, but what about the other papers, like *my* paper, are they leftovers?".

Looking ahead now: DISC 2026 will take place in Rome, and the PC will be chaired by Keren Censor-Hillel.

For DISC 2027, two strong bids were submitted for Lisbon and Wroclaw, and the conference will likely be held in one of these cities. There were discussions about the carbon footprint of these locations, a topic that has gained importance at DISC in recent years. While both cities are in Western Europe, where much of the community is based, neither is easily accessible by train for most attendees. (In particular I was surprised to learn that traveling from Madrid to Lisbon by train already takes 10 hours!)

There were also two more exploratory bids. One bid came from our group in Lyon and Mikael Rabie's group in Paris, proposing to host the conference in either city. That bid included a condition: allowing a portion of the talks to be presented online to help reduce the conference's carbon footprint. The other bid was for Kathmandu — a less conventional location for a TCS conference. However, it successfully hosted SSS last year, and attendees provided positive feedback.

Random bits and corridor chats Let me finish with a list of random topics:

- At the lunch breaks, many of us discovered the concept of butter machine (a.k.a. butter dispenser), which seems to be standard in Germany: a device where you place a plate or slice of bread in a designated spot, and it dispenses a dose of butter!

- Tijn de Vos really wanted to have question for his talk at the ADGA workshop. So every slide would start with a question, and Tijn would not continue until someone in the audience had asked that question. It worked extremely well!

- The name tags were printed on both sides, which was great to avoid asking people to turn their tags around to see their names.

- There were multiple discussions about chatbots: how they were introduced in the STOC submission process, how they can or cannot help with research, how they could help formalization of proofs (in Lean and such).
- Someone used the "ϒ" Greek letter in a talk, possibly the only Greek letter I'd never encountered in a scientific context before!

Many thanks to organizing team I almost forgot to thank the organizing team! I guess this is because the conference was perfectly seamless: you even forget that there are organizers! More seriously, many thanks to Stefan and Joel, and the rest of the organizing team: Yannic Maus (workshop chair), William K. Moses Jr. (publicity Chair), Birgit Hohmeier-Touré, Samar Khaksar, Darya Melnyk, Tijana Milentijevic, Julien Dallot, Dorian Gutschenreiter, Jakob Solnerzik, Olivier Stietel, Robin Vacus, and Yaroslav Verbitsky.

References

[1] Michel Raynal and Nicola Santoro: A Scientific Story (1985-2025) from WDAG to DISC *Bull. EATCS*, 147, 2025. URL: https://bulletin.eatcs.org/index.php/beatcs/article/view/860

[2] Moni Naor and Larry J. Stockmeyer: What Can be Computed Locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995. DOI: https://doi.org/10.1137/S0097539793254571

The Computational Complexity Column

by

Michal Koucký

Computer Science Institute, Charles University
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
koucky@iuuk.mff.cuni.cz
https://iuuk.mff.cuni.cz/~koucky/

BEATCS no 148

Using Hardness vs Randomness to Design Low-Space Algorithms

Edward Pyne*
MIT
epyne@mit.edu

Roei Tell†
University of Toronto
roei@cs.toronto.edu

Abstract

Can we use "hardness vs randomness" techniques to design low-space algorithms? This text surveys a sequence of recent works showing ways to do that. These works designed algorithms for certified derandomization and for catalytic computation (which work unconditionally), derandomization and isolation algorithms from remarkably mild assumptions, and "win-win" pairs of algorithms that, for every input, solve either derandomization or another important problem (e.g., *s-t* connectivity) on the input.

Underlying these constructions are new, specialized "hardness vs randomness" tools for the setting of low-space algorithms. We describe these technical tools, most notably constructions of pseudorandom generators whose reconstruction algorithm (i.e., the security reduction) is a deterministic low-space algorithm. We also explain a key part of obtaining deterministic reconstruction, which is deterministic transformations of distinguishers to bit-predictors.

We pose a host of open questions that explore new ways of using hardness vs randomness to design low-space algorithms. These questions address problems in derandomization, catalytic computation, explicit constructions, learning algorithms, and more.

*Supported by an NSF Graduate Research Fellowship.
†Supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant RGPIN-2024-04490.

Contents

1 Hardness vs Randomness when Memory is Scarce

One of the most influential techniques in theoretical computer science is the "hardness vs randomness" paradigm. Originating in cryptography [10, 87], this technique transforms lemons into lemonade: It takes hard problems, which no efficient algorithm can solve, and uses them to design algorithms for other problems. Perhaps the most well-known example is the Nisan-Wigderson generator [58], which (combined with additional tools by Impagliazzo and Wigderson [46]) yields the following textbook result:

Theorem 1.1 ([46])**.** *If* $\mathbf{E} = \mathbf{TIME}[2^{O(n)}]$ *has a problem hard for circuits of size* $2^{.01 \cdot n}$ *on all input lengths, then there is a deterministic polytime algorithm that, given a circuit* $D : \{0,1\}^n \to \{0,1\}$*, estimates* $\mathbb{E}[D]$ *up to an additive error of* $1/n$*.*

Since the problem of estimating the expectation of a given circuit is complete for **prBPP**, the conclusion in Theorem 1.1 implies that **prBPP** = **prP**.

Over the three decades since the underlying tools were developed, they have been used to *design other types of algorithms*. And most notably, many of the algorithms designed using these tools are *unconditional*, i.e. do not actually need to start from a hardness assumption! Among other things, hardness vs randomness tools have been used to unconditionally obtain constructions of randomness extractors, condensers, and expander graphs [45, 75, 76], learning algorithms [12], pseudodeterministic explicit constructions [15], circuit lower bounds [13], and worst-case to average-case reductions [43].

Can we design low-space algorithms using hardness vs randomness? Early on, Klivans and van Melkebeek [49] showed that hardness vs randomness tools can be adapted to yield low-space algorithms. In particular, under a stronger hardness assumption than in Theorem 1.1 – a hard problem in $\mathbf{SPACE}[O(n)]$ rather than only in $\mathbf{E} \supseteq \mathbf{SPACE}[O(n)]$ – the concluded algorithm can run in *logarithmic space*:

Theorem 1.2 ([49])**.** *If* $\mathbf{SPACE}[O(n)]$ *has a problem hard for circuits of size* $2^{.01 \cdot n}$ *on all input lengths, then there is a deterministic logspace algorithm that, given a circuit* $D : \{0,1\}^n \to \{0,1\}$ *that can be evaluated in space* $O(\log n)$*, estimates* $\mathbb{E}[D]$ *up to an additive error of* $1/n$*.*

Theorem 1.2, however, had less follow-ups compared to Theorem 1.1. In fact, we are not aware of *any* work prior to 2023 that used hardness vs randomness to design logspace algorithms. Why is this?

One explanation is that we didn't know how to exploit the fact that when designing low-space algorithms, we are usually trying to solve potentially easier problems, compared to the conclusion of Theorems 1.1 and 1.2. The untapped potential is that when tackling easier problems, one may hope to weaken the hardness assumption, in which case (using the many existing ideas) we could eventually obtain unconditional algorithms. For example, consider the problem of derandomizing probabilistic logspace machines. This problem reduces to estimating the expectation of a Read-Once Branching Program (ROBP),[1] a weak computational model for which exponential lower bounds are well-known (see, e.g., [9]). However, past applications of hardness vs randomness to this problem made little use of this fact (see [30, Section 1.1]).

[1] This is because the procedure $B_x(\cdot)$ obtained by fixing an input x to the machine (i.e., B_x takes the randomness r as its input) is an ROBP.

The good news. The main focus of this text is a sequence of recent works [28–30, 52, 60, 61] that used hardness vs randomness tools to design low-space algorithms for important problems – including derandomization, *s*-*t* connectivity, function composition, and catalytic computation – where some of these algorithms work unconditionally and others work under mild assumptions.

A key bottleneck to get through was resolving the issue above, i.e. figuring out that there *is* a critical feature of what we call "the logspace setting" (i.e., of various relaxed problems that we try to solve using logspace algorithms) that hardness to randomness tools can exploit.[2] Another facilitator of these works is recent developments in hardness vs randomness more generally (i.e., for algorithms that aren't necessarily low-space; see, e.g., the survey [24]), and in particular a sequence of new technical tools for the time-bounded setting, which the works above migrated and adapted to obtain new space-bounded tools.

What's in this text? The goal of this survey is to explain three things:

1. **How to exploit the logspace setting.** In Section 2 we will present the main new technical feature of hardness vs randomness tools that can be obtained in the logspace setting (and also possibly beyond that), which is called deterministic reconstruction.

2. **The new algorithms.** In Section 3, we present low-space algorithms from recent years that use hardness vs randomness, and in particular that exploit deterministic reconstruction. Specifically, we will mention certified derandomization, catalytic algorithms, win-win pairs of algorithms (for *s*-*t* connectivity and for efficient copmosition), and derandomization of **BPL** from remarkably weak assumptions.

3. **Underlying technical tools: New reconstructive generators.** Finally, in Section 4, we explain how to achieve deterministic reconstruction: first in the special case of trying to prove **BPL** = **L**, and then for several other problems mentioned above. We will introduce a concept called distinguish-to-predict transformations, and describe some new pseudorandom generators in more detail.

In Section 5 we suggest a host of interesting and potentially tractable open problems, which explore new ways of using hardness vs randomness to design low-space algorithms.

Finally, an extra goodie of these developments is elegant proofs for several classical results, which use the concepts introduced above; we present these proofs in Appendix A.

2 Exploiting the Logspace Setting: Deterministic Reconstruction

To explain the new feature of hardness vs randomness tools that can be achieved in the logspace setting, let us recall how these tools work. The "recipe" behind them looks roughly like the following.

A hardness vs randomness recipe. There is a string $f \in \{0,1\}^T$, which we call the hard string, that we hope (or assume) is a truth table that cannot be computed by circuits of size $s \ll T$.[3]

[2] For the special case of derandomizing probabilistic logspace machines (i.e., trying to prove **BPL** = **L**), the technical proof of this is simple and accessible – we encourage readers to check out Sections 4.1.2 and 4.2.1!

[3] By which we mean there is no size-s circuit C that, given $j \in [T]$, outputs f_j. If there is such a circuit C, we say its truth table is f, or that it computes f.

We give f to a deterministic algorithm GEN that has the following behavior. Given a circuit D, the algorithm $\mathsf{GEN}^f(D)$ outputs an estimate for $\mathbb{E}[D]$. If this estimate is good (say, within an additive factor of $1/6$ of the true expectation), then we solved an interesting problem (i.e., deterministically estimated D's acceptance probability). Otherwise, the proofs of correctness show that there exists a circuit C with the following properties:

- The size of C is less than s, and C makes queries to D.
- The truth-table of C^D is f.

Thus, if GEN^f fails to estimate $\mathbb{E}[D]$ well, and D is a small circuit (of size less than s), then there exists a small circuit for f, namely C^D. So assuming f has no size-s circuits, the derandomization must succeed.

The complexity of reconstruction. The recipe above shows that when GEN fails, f can be compressed, to the form of a small circuit. Being able to compress a given string f is potentially useful, but:

Where does this circuit C come from? Can we find it?

To formalize this question, we extend our recipe to a pair of algorithms $(\mathsf{GEN}, \mathsf{REC})$.

- The generator $\mathsf{GEN}^f(D)$ works as before, but if it does a bad job...
- ... the reconstruction algorithm $\mathsf{REC}^f(D)$ prints a small circuit C such that C^D computes f.[4]

In [46, 58], the reconstruction REC is non-explicit and inefficient. However, it was quickly realized [47] that in certain settings we can also design *efficient, probabilistic* reconstruction algorithms; for example, recent works showed that this is true when f is computable by uniform low-depth circuits (this line-of-work is often referred to as "uniform hardness vs randomness"; see, e.g., [21, 22, 47, 77]).

The complexity of the reconstruction REC turns out to be more important than how it may initially seem. This is because the recipe above is used in non-obvious ways, some of which yield *unconditional* algorithms. Indeed, in these applications REC plays several roles in disguise; for example:

1. When the recipe is used for conditional results in the straightforward way (as in, say, [22, 46, 47, 49, 58]), the complexity of REC determines the hardness assumption.
2. The recipe can be instantiated with an *intentionally faulty* GEN (i.e., in a setting where GEN is guaranteed to fail), in which case REC is the actual algorithm, and it unconditionally works. In this case, the complexity of REC is the complexity of the final algorithm (see, e.g., [12, 43, 76]).
3. The recipe can also be instantiated with both GEN and REC trying to solve the same problem (!). In this case, the guarantee that at least one of them works means that we can unconditionally solve the problem, and the complexity of our algorithm is the worst of both worlds (i.e., the maximum among the complexity of GEN and of REC; see, e.g., [13, 15, 45]).

An important open problem is extending the class of functions for which we can get an efficient probabilistic REC (see, e.g., open problems in [22, 25]). We will now ask about another direction.

[4] In cryptography the reconstruction algorithm is usually called "the security reduction".

Can we hope for a generator with deterministic reconstruction? That is, can we hope to design $(\mathsf{GEN}, \mathsf{REC})$ such that for every f, whenever the generator $\mathsf{GEN}^f(D)$ fails, the reconstruction $\mathsf{REC}^f(D)$ efficiently and *deterministically* compresses f to a small circuit?

This is not a pipe dream: such a construction trivially exists if **prBPP** = **prP** (because then there is GEN that simply ignores f and estimates $\mathbb{E}[D]$ correctly). However, for general D's, this might be too good to be proved at the moment, since it would imply that **prBPP** = **prZPP**. It is instructive to see why. Consider an algorithm that gets input D and wants to estimate $\mathbb{E}[D]$. Given $(\mathsf{GEN}, \mathsf{REC})$ with deterministic reconstruction, the algorithm can generate a random f, run $\mathsf{REC}^f(D)$, and if $\mathsf{REC}^f(D)$ fails to print a circuit for f, the algorithm uses $\mathsf{GEN}^f(D)$ to estimate $\mathbb{E}[D]$. Indeed, whenever REC fails, the algorithm can be certain that $\mathsf{GEN}^f(D)$ succeeds, and REC fails for most f's, since they are incompressible.[5]

Thus, deterministic reconstruction isn't just a way of transforming hypothesized hardness into randomness – it is also unconditional derandomization by itself (for a relevant class of D's).

> The main technical discovery opening the door for the recent line-of-works, from [60], is that for *restricted classes of D's* that arise when studying problems in the logspace setting, we can design $(\mathsf{GEN}, \mathsf{REC})$ whose reconstruction REC is deterministic. Moreover, both GEN and REC can be *logspace* algorithms, i.e. running in deterministic space $O(\log |f|)$.

In fact, there are by now several constructions of generators with deterministic reconstruction for restricted classes of D's. We survey some known constructions and explain which classes of D's they work for in Section 4. For now, we encourage the reader to think of the simplest case, which is when D is a Read-Once Branching Program (ROBP); as mentioned in Section 1, ROBPs arise when trying to prove **BPL** = **L**.

Remark 2.1. In the recipe above, we may also allow f to be a hard string that depends on D (and some generators presented in this survey will do so). In this setting it is more convenient to think of both $D = D_x$ and $f = f(x)$ as functions of an input x. That is, GEN_f gets input x tries to estimate $\mathbb{E}[D_x]$ by computing $f(x)$; and if it fails, a reconstruction $\mathsf{REC}_f(x)$ computes $f(x)$ too efficiently (even a small circuit whose truth-table is $f(x)$). This type of generator was introduced by Goldreich [34, 36], who called it a targeted generator (where x is the "target"). We elaborate on this in Section 4.3.

3 New Low-Space Algorithms

Now we have our deterministic reconstruction hammer, what nails can we hit with it? We first cover a straightforward application that we call certified derandomization. However, as is usually the case with hardness to randomness, there are many non-obvious applications which we will cover subsequently. As we go, we provide forward pointers to the technical tools (i.e., to the generators and reconstruction algorithms presented in Section 4) required for each application.

3.1 Certified Derandomization

Recall that Theorem 1.2 states that, assuming $\mathbf{SPACE}[O(n)]$ is hard for circuits of size $2^{.01 \cdot n}$, then **BPL** = **L**. But what if the assumption is false? In that case, the derandomized algorithm

[5] In fact, a slightly more general notion of "deterministic reconstruction" is equivalent to **prBPP** = **prZPP**; see [52].

for **BPL** will fail silently: it will still return a value, but this value has no correctness guarantee. With deterministic reconstruction, we can do better:

Theorem 3.1 ([60])**.** *Let $\mathcal{L}_{hard} \in$ **SPACE**$[O(n)]$. For every $\varepsilon > 0$ and $L \in$ **BPL** there is a deterministic logspace algorithm $\mathcal{A}$ that on input $x \in \{0,1\}^n$, either:*

1. *Prints L_x.*
2. *Outputs "I cannot produce an output because the hardness assumption is false", followed by a circuit C of size $2^{\varepsilon n'}$ for $\mathcal{L}_{hard}$ on inputs of size $n' = \Theta(\log n)$.*

We briefly sketch how this is a consequence of deterministic reconstruction. We try to use GEN^f to derandomize the **BPL** machine on x, where f is the truth-table of $\mathcal{L}_{\text{hard}}$ on n'-bit inputs. But before trusting the output of GEN^f, we first determine if REC^f outputs a small circuit for f, and if so we return that circuit; otherwise, we use the output of GEN^f to derandomize as usual. If the hardness assumption is true, no such small circuit exists, and hence the former situation will never occur, and hence our algorithm will always derandomize L in logspace. But if we are wrong and there is such a small circuit, we either get a certificate of this fact (i.e., the circuit that REC^f produces), or are guaranteed that GEN^f works nonetheless!

This result has something of a "win-win" favor (either we derandomize or we get a small circuit for f), which will come up again in subsequent results.

3.2 Catalytic algorithms

How valuable is a full memory? In the setting of catalytic algorithms, we augment a standard logspace algorithm with a much larger catalytic space. This space has an arbitrary initial configuration, which can be edited during the computation but must be restored to the initial configuration at the end of the computation. Introduced in 2014 by Buhrman et. al. [6], the model has played a central role in recent breakthrough results, such as the Tree Evaluation algorithm of Cook and Mertz [17,18] and the simulation of time-T computation in space $\tilde{O}(\sqrt{T})$ by Williams [85].

A main technique to build catalytic algorithms (i.e., algorithms that utilize catalytic space) was dubbed "compress or random" by Mertz [54]. This technique starts with some dense property $\mathcal{P}$ of strings that is useful for designing an algorithm (e.g., being a hard truth-table). For the catalytic algorithm, if the initial content τ of its catalytic tape satisfies $\mathcal{P}$, the algorithm can just use τ. However, if $\tau \notin \mathcal{P}$, then the contents τ of the catalytic space lies in some small set of strings (i.e., the complement of $\mathcal{P}$), and hence the content τ is – at least information-theoretically – compressible! If we could algorithmically compress τ in this case, we would unconditionally design an algorithm by a win-win analysis: either the good property holds, or we free up a large amount of space and solve the problem by brute force.

But how do we implement this technique? The main challenge is that we need a deterministic, low-space algorithm that compresses τ when τ is bad. Indeed, now comes the punch-line – this is exactly what deterministic reconstruction provides! Hence, hardness vs randomness tools with deterministic reconstructions are useful for designing catalytic algorithms.

Two algorithms, and a possibility. First, a basic open problem about probabilistic logspace is search-to-decision reductions. Specifically, an algorithm for the *decision* problem of derandomization (i.e., decide whether $\mathbb{E}[D]$ is high or low) is currently not known to imply an algorithm of similar space complexity for the corresponding *search* problem (i.e., print a distribution that is pseudorandom for D).

For catalytic algorithms, however, this was recently proved to be true – indeed, using "compress or random" with a generator that has deterministic reconstruction. In particular, recall that **CL** denotes the class of problems solvable with logarithmic space and polynomial catalytic space; then:

Theorem 3.2 (Informal, see Theorem 5.4 [52])**.** *Let $\mathcal{D}$ be a class of circuits evaluable in* **CL***, and suppose that there is a* **CL** *algorithm that estimates $\mathbb{E}[D]$ for $D \in \mathcal{D}$ up to additive error $1/10n$. Then there is a* **CL** *algorithm that, given D, prints a distribution that $(1/3)$-fools D.*

A second catalytic algorithm using this technique simulates **BPL** in **CL**. This result was already shown in the first paper about catalytic computing [6], relying on algebraic techniques and on the somewhat indirect route "**BPL** $\subseteq$ logspace-uniform-$\mathbf{TC}^1$ $\subseteq$ **CL**". A more direct algorithm simulating **BPL** in **CL** was recently shown, using "compress or random" and a generator with deterministic reconstruction.

Theorem 3.3 ([6], an alternative proof in [28])**.** *We have that* **BPL** $\subseteq$ **CL**.

Another alternative proof of Theorem 3.3 follows from the fact, proved recently in [14,48,62], that randomized catalytic space coincides with deterministic catalytic space (i.e., that **CBPL** = **CL**; the last of these works even showed that non-deterministic catalytic algorithms can be simulated by deterministic ones). The original algorithm in [14] was, indeed, a "compress-or-random" algorithm using a generator with deterministic reconstruction. However, in subsequent work [48] a simpler algorithm was shown, using a compression technique that does not involve hardness-vs-randomness.

We believe that deterministic reconstruction still has a major role to play in catalytic algorithms, because it gives a highly generic compression technique. We suggest this as an open problem in Section 5.

3.3 Win-win pairs of algorithms

As mentioned in Section 2, hardness vs randomness can turn lemons into lemonade, by turning the *non*-existence of efficient algorithms for one problem into an efficient algorithm for another problem.

Interestingly, recent works [29, 61] go further than that, by showing an "instance-wise" win-win between derandomization and two fundamental problems in space complexity. Specifically, they showed that for every fixed input x, either we can derandomize **BPL** on x, or we can solve a fundamental problem in space complexity on x (i.e., much more efficiently than the best currently known algorithm for that problem).

3.3.1 Graph connectivity

First, consider the problem of s-t-connectivity, where we are given a directed graph $G = (V, E)$ on n vertices and two vertices s, t, and need to decide if there is a path from s to t.

This problem is complete for nondeterministic logspace (**NL**), and can be solved in space $O(\log^2 n)$ via the famous result of Savitch [66]. However, Savitch's algorithm runs in time $2^{O(\log^2 n)} \gg \text{poly}(n)$. The problem can also be solved in time $O(n^2)$ and space $O(n)$, via a simple Breadth First Search. But what about a time *and* space efficient solution? Whether such an algorithm exists is wide open!

A result of Barnes et. al. [5] solves s-t-connectivity in polynomial time and space $O(n/2^{\sqrt{\log n}}) = o(n)$, but even the following is up in the air:

Question 3.4. Is there a constant $\varepsilon > 0$ and an s-t-connectivity algorithm running in simultaneous time $n^{\log^{1-\varepsilon} n}$ and space $n^{1-\varepsilon}$?

In a restricted computational model that captures all known s-t-connectivity algorithms, an algorithm with these parameters matching Question 3.4 is known not to exist [20, 53, 59]. However, this restriced model does not capture general algorithms (cf., e.g., the difference between [19] and [18]).

A win-win pair of algorithms. If one is pessimistic, and believes that there isn't an algorithm as in Question 3.4, what does that imply? A natural hope is to use randomness vs randomness to build a useful algorithm from this hardness. Classical hardness vs randomness yields non-uniform circuits (or, in some settings, probabilistic algorithms), but with deterministic reconstruction we can do better.

Specifically, a recent result of [29] ties together s-t connectivity with the problem of estimating random walk probabilities (which is complete for **BPL**): For *every* given graph G, at least one of these problems can be solved on G much more efficiently than the best currently known algorithm.

Theorem 3.5 ([29, Theorem 1])**.** *There are algorithms $\mathcal{A}_1, \mathcal{A}_2$ such that for every graph G on n vertices, one of the following holds:*

- *$\mathcal{A}_1(G)$ solves s-t connectivity in G in polynomial time and polylogarithmic space.*
- *$\mathcal{A}_2(G)$ estimates length-n random walk probabilities in G in non-deterministic logspace.*[6]

Moreover, both algorithms report if they fail to compute the desired answer, and do not exceed their resource bounds in any case.

How is deterministic reconstruction used? The use of deterministic reconstruction in the proof of Theorem 3.5 is not straightforward, and proceeds roughly as follows. Consider the following algorithm for computing s-t-connectivity in small time and space. First, compute all 1-step connectivity information (i.e. for every vertex pair u, v, decide if there exists a 1-step path from u to v), and write this down in a matrix $M_1 \in \{0,1\}^{n \times n}$. But rather than storing M_1 on the worktape using n^2 bits, try to *compress* M_1 to a circuit C_1 of size polylog(n). If this compression step fails, abort. Otherwise, compute the 2-step connectivity information M_2 (which is easy to do given M_1), delete C_1, and compress M_2 into C_2. After n iterations, we obtain a compressed encoding of M_n, which holds the connectivity information of the graph.

But what compression algorithm can we use? The key idea of [29] is to use a deterministic reconstruction algorithm REC as our compression algorithm in each iteration i, with $f = M_i$ as the hard string. When the compression *fails* it must be the case GEN^{M_i} succeeds in estimating random walk probabilities. Since we can produce each matrix M_i in **NL**, either all the compression algorithms work (and we solve connectivity quickly and in low space) or we obtain a *nondeterministic* derandomization of walks on G.

In [29, 61] they showed that this approach can be instantiated in a more general way, yielding a new targeted generator for the logspace setting. We elaborate on this in Section 4.3.

[6] The estimation is up to an additive $1/\operatorname{poly}(n)$ error. The algorithm runs in logspace, makes non-deterministic guesses, and either declares *fail* if the guess sequence is bad, or outputs a single canonical matrix only depends on the graph G, or outputs a special symbol $\perp$ indicating that $\mathcal{A}_2$ does not succeed on G (in which case $\mathcal{A}_1$ succeeds on G).

A win-win for complexity classes. Theorem 3.5 gives a pair of algorithms such that, for every input G, at least one of these algorithms works. Working in a scaled-up regime and with different parameters (but with the same fundamental idea), in [29] they deduced a win-win for classical complexity classes.

Theorem 3.6 ([29, Theorem 2])**.** *For every constant $\varepsilon > 0$, at least one of the following holds:*

- $\mathbf{NSPACE}[n] \subseteq i.o.\mathbf{TISP}\left[2^{O(n^{2-\varepsilon})}, n^{O(1)}\right]$.
- $\mathbf{BPSPACE}[n] \subseteq \mathbf{SPACE}\left[O(n^{1+\varepsilon})\right]$.

Note that the second case in Theorem 3.6 does better than a simple scale-up of the second case in Theorem 3.5, because the simulation of **BPSPACE** (i.e., the derandomization) does not use non-determinism.

3.3.2 Composing low-space algorithms

Consider the following fundamental question: what is the complexity of *composing* low-space algorithms? (Jumping ahead, we will explain how this question was recently attacked using hardness vs randomness.)

Given a function $f : \{0,1\}^n \to \{0,1\}^n$ computable in time $T \geq n$ and $S = O(\log n)$, there are two basic ways to compute the function $x \to f(f(x))$. First, a naive strategy: compute $y = f(x)$, store it in the workspace, then compute $f(y)$. This method runs in time $O(T)$, but space at least $n \gg S$ due to needing to store y. This space overhead means that essentially every low-space algorithm in theoretical computer science makes use of a second option, coined "emulative composition" by Goldreich [33], or "composition of space-bounded algorithms". Here, we begin to compute $f(y)$, and each time the algorithm queries a bit of y, we recompute it on the fly. This reduces the space to $O(S)$, but quadratically blows up the time, i.e. to T^2 instead of T. One may also interpolate between the two methods using sub-quadratic time and super-logarithmic space, as long as the time-space product is at least $\tilde{O}(T^2 \cdot S)$ (see [61, Proposition 4.1]).

A seemingly natural question is whether we can get the best of both worlds. That is, can we compose algorithms in low space without paying a significant time overhead? It turns out that for linear time algorithms, i.e. $T(n) = O(n)$, the answer is *unconditionally* no:

Theorem 3.7 ([86], appearing in [61, Theorem 1.2])**.** *There is a length-preserving function $f : \{0,1\}^* \to \{0,1\}^*$ computable in simultaneous linear time and logspace, where any algorithm computing $x \to f(f(x))$ must have time-space product at least $n^{1.33}$.*

Thus, for linear time, composition incurs an unavoidable time-space overhead. An obvious next step is to ask if this lower bound holds also for large polynomial time bounds T. In [61] they showed that proving such a statement implies average-case derandomization of logspace:

Theorem 3.8 ([61, Theorem 1.3])**.** *Suppose that there is $\delta > 0$ such that for every polynomial $T(n)$ and constant $\varepsilon > 0$ the following holds. There is a function f computable in time T and space $O(\log n)$ such that any algorithm computing $f(f(x))$ successfully on an ε-fraction of inputs $x \in \{0,1\}^n$ requires time-space product $T^{1+\delta}$.*[7] *Then $\mathbf{BPL} \subseteq \cap_{\varepsilon>0}\mathbf{avg}_\varepsilon\mathbf{L}$.*

Theorem 3.8 was also extended in [61] to the setting of composing f for k times (i.e., asking whether low-space algorithms require time $T^{\Omega(k)}$), and to the scaled-up setting, in which we focus on worst-case composition and derandomization (rather than average-case).

[7] In this statement we refer to the runtime of both $f(x)$ and $f(f(x))$ as a function $T = T(n)$ of the length of the input $x \in \{0,1\}^n$ to the composition. We could alternatively describe f_1 as running in time T and outputting T bits, and describe f_2 as running in linear time (in T).

A word about the technique. The strategy for proving Theorem 3.8 is to again construct a win-win pair of algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$, as follows: For every x, either $\mathcal{A}_1(x)$ outputs $f(f(x))$ in small time and space, or $\mathcal{A}_2(x)$ simulates the (predetermined) **BPL** machine on input x.

The algorithms are designed in similar fashion to the win-win pair in Theorem 3.5, except that the generator now does not use the i-step connectivity matrices (for $i = 1, ..., n$) as a source of hardness, but instead uses the strings $f(x), f(f(x)), ...$ obtained by repeatedly computing f. (Indeed, in the version in Theorem 3.8 it uses the two strings $f(x)$ and $f(f(x))$, and when considering hardness of k-wise composition, it uses $f^{(i)}(x)$ for $i = 1, ..., k$). The main challenge comes in the reconstruction argument: whenever the generator fails, we want to compute the last layer $f^{(k)}(x)$ using near-linear time and polylogarithmic space. The technical tools described so far don't seem to suffice for this purpose, since the reconstruction runs in large poly(T) time. In [61] they showed a new (GEN, REC) pair (i.e., a generator and deterministic reconstruction) wherein REC runs in near-linear time and polylogarithmic space, and is also read-once over its input (i.e., reads its input in streaming fashion, bit-by-bit in order), which is another feature needed to avoid time blow-ups when implementing the idea above. This is described in detail in Section 4.3.2.

3.4 Derandomization and isolation: BPL = L using hardness vs randomness?

The question of **BPL** = **L** has been studied for decades via combinatorial constructions of pseudorandom generators for ROBPs and related models (see, e.g., the surveys [44, 65]). Can we instead attack this question using hardness vs randomness? The suggestion in [28] is as follows:

> Reduce **BPL** = **L** (or relaxed versions of it) to lower bounds that are so weak that we can unconditionally prove them.

Their idea was to rely on deterministic reconstruction to reduce derandomization to lower bounds for *uniform, deterministic algorithms*, and in particular for classes of weak uniform deterministic circuits, in which case proving these lower bounds seems tractable. (For context, recall that versions of **BPP** = **P** are known to follow from certain hardness for uniform *probabilistic* algorithms [22, 47, 51, 77].)

3.4.1 Derandomization from remarkably weak lower bounds

We present two appealing instantiations of this approach, from [28] and [61] (improving on a prior result from [29]), which reduce a relaxed version of **BPL** = **L** to remarkably weak hardness assumptions. The conclusions below will be derandomization of linear space (rather than logspace), and the assumptions will be extensions of unconditionally known lower bounds.

Theorem 3.9 ([28, Theorem 2])**.** *Suppose that* **SPACE**$[O(n)]$ *is hard for* **SPACE**$[C \cdot n]$*-uniform* $(\mathbf{TC}^0)^{\mathsf{ROBP}}$ *circuits of size* $2^{\varepsilon \cdot n}$*,*[8] *for some* $\varepsilon > 0$ *and all* $C > 1$*. Then,* **BPSPACE**$[O(n)] \subseteq$ **i.o.SPACE**$[O(n)]$.

The assumption in Theorem 3.9 "almost" follows from a simple diagonalization argument, where the only problem is that the circuit in the lower bound is printed by an algorithm using

[8] The notation $(\mathbf{TC}^0)^{\mathsf{ROBP}}$ refers to $\mathbf{TC}^0$ circuits with oracle access to ROBPs, where the size bound $2^{\varepsilon \cdot n}$ accounts for the total description size of the circuit and of the ROBP together.

space $C \cdot n$, whereas the upper bound uses space $c \cdot n$, where $c < C$. This obstacle is not insurmountable. In fact, a scaled-down version of the hypothesis in Theorem 3.9 (referring to logspace and to polynomial-sized circuits) is unconditionally known, using the techniques of Santhanam and Williams [73] (see [28, Proposition 1.2]). Thus, scaling up the known lower bound would suffice to prove that $\mathbf{BPSPACE}[O(n)] \subseteq \mathbf{i.o.SPACE}[O(n)]$.[9]

Theorem 3.10 ([61, Theorem 1.6])**.** *There is $c > 1$ such that the following holds. Suppose that the following is true:*

> *There is an algorithm that gets input 1^n, runs in space $O(\log n)$, and outputs a list of n-bit strings $f_{n,1}, ..., f_{n,m}$ such that every* deterministic uniform one-pass streaming algorithm *that runs in space $(\log n)^c$ and time n^c fails to compress $f_{n,i}$ to size* $\mathrm{polylog}(n)$*, for some i.*[10]

Then, $\mathbf{BPSPACE}[n] \subseteq \mathbf{SPACE}[O(n)]$.

The assumption in Theorem 3.10 is appealing because it refers to a very weak computational model, and in particular, because the reduction of derandomization to a lower bound does not incur overhead in the computational model. To see this, recall that deterministic uniform one-pass streaming algorithms are essentially equivalent to uniform deterministic ROBPs.[11] Thus, Theorem 3.10 reduces estimating the acceptance probability of ROBPs to a lower bound for ROBPs (and moreover, for uniform deterministic ROBPs). This is particularly striking, because exponential lower bounds for ROBPs, even non-uniform ones, have been well-known for decades (see, e.g., [9]). The lower bound needed in Theorem 3.10 differs from what is known because it refers to hardness of compressing an explicit string (by ROBPs of size quasipolynomial in the string's length), rather than to hardness of computing an explicit function (by ROBPs of size that is smaller than the function's truth-table).

Technically, at a high level the two results use hardness vs randomness in the straightforward way (i.e., the hardness assumption is that the reconstruction algorithm fails), and the main challenge is obtaining very efficient reconstruction algorithms. For example, in Theorem 3.10 the reconstruction algorithm is a small uniform deterministic ROBP (which tries to compress the $f_{n,i}$'s), rather than a general algorithm. The underlying technical tools are presented in Sections 4.1 and 4.2.2.

3.4.2 Derandomization with minimal memory footprint

The hardness to randomness paradigm allows us to deduce derandomization, but what is the precise cost of derandomization? Recall that classical conditional results [46, 58] transform any randomized linear time algorithm into a deterministic algorithm running in time that is polynomial but that may be very large (i.e., n^c for an unspecified constant $c \gg 1$).

Doron *et al.* [27] posed the question of whether it is possible to simulate all randomized algorithms while incurring minimal time overhead; ideally, we want to incur no time overhead

[9] A variation on Theorem 3.9 was shown in [61, Theorem 7.10], in which the circuits in the lower bound are only of polynomial size (rather than size $2^{\varepsilon \cdot n}$), but they are general circuits (rather than $(\mathbf{TC}^0)^{\mathsf{ROBP}}$ circuits).

[10] The notion of "compressing" here means outputting a machine M of size $\mathrm{polylog}(n)$ that gets input $i \in [n]$, runs in space $\mathrm{polylog}(n)$ and time $\mathrm{poly}(n)$, and outputs $(f_n)_i$.

[11] The main difference between the two is that for uniform ROBPs it is more natural to specify their uniformity (e.g., the ROBP is logspace-uniform), whereas for streaming algorithms it is more natural to specify their computational complexity (e.g., the algorithm runs in polynomial time). The technical result in [61] refers to a model in which both restrictions simultaneously apply: as an ROBP it is logspace-uniform, and as a streaming algorithm its running time is at most n^c.

at all. Their work and follow-ups [7, 22, 23, 72] showed that, under strong hardness assumption, derandomization with very small time overhead is indeed possible.[12]

What about derandomizing with minimal space overhead? This question was posed in [30], which asked whether we can simulate probabilistic algorithms running in space S by deterministic algorithms running in space $c \cdot S$ for $c \geq 1$ that is as small as possible (ideally $c \approx 1$).

Loosely speaking, their original work deduced this with $c \approx 2$ under two assumptions: very efficient cryptographic PRGs, and the existence of a logspace algorithm printing strings that are hard to probabilistically compress.[13] Subsequently, in [29] they showed that these assumptions can be replaced by a single, non-cryptographic hardness assumption, for uniform deterministic machines.

Theorem 3.11 (Informal, see Theorem 5 [29])**.** *Assume for every C there exists a function f mapping n bits to n^2 bits that is computable in space $(C+1) \cdot \log(n)$, but no deterministic, low-space algorithm R can achieve the following. On input x, R print a machine M of description size $O(n)$ where M runs in space $C \cdot \log(n)$ and prints $f(x)$. Then, for any $S(n) = \Omega(\log n)$ and constant $\varepsilon > 0$,*

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{SPACE}[(2+\varepsilon) \cdot S].$$

The high-level proof approach for Theorem 3.11 is similar to that underlying Theorems 3.9 and 3.10, but the technical details are considerably more complicated, since this result needs to get a generator that is computable with essentially no space overhead, while still maintaining a deterministic reconstruction. In [29] they showed this by composing a reconstructive generator with the Forbes-Kelley PRG [31], and building a specialized deterministic reconstruction for the latter; see Section 4.2.2 for details.

3.4.3 Isolation from remarkably weak lower bounds

Finally, consider the question of making nondeterministic algorithms *unambiguous*, i.e. having at most a single convincing witness. In the setting of time-bounded algorithms, the well-known result of Valiant and Varizani [82] gives a randomized reduction from **NP** to **prUP** (recall **UP** is the class of problems with a single valid witness for every YES instance, and **prUP** is the corresponding class of promise problems). However, assuming $\mathbf{NP} \not\subset \mathbf{P}/\mathrm{poly}$, there is no *deterministic* algorithm that reduces **NP** to **prUP** by "isolating" witnesses, i.e. by mapping circuits with many satisfying assignments to circuits that reject all but exactly one of these assignments [26].

Perhaps surprisingly, in the setting of space-bounded algorithms there is evidence that we *can* deterministically turn all nondeterministic algorithms into unambiguous ones, given by Wigderson, Gál, Allender, Reinhardt, and Zhou:

Theorem 3.12 ([4, 40, 63, 83])**.** *Suppose that $\mathbf{SPACE}[O(n)]$ is hard for circuits of size $2^{\varepsilon n}$ on all input lengths, for some $\varepsilon > 0$. Then $\mathbf{NL} = \mathbf{UL}$.*

[12] Specifically, these works conditionally deduced worst-case derandomization with a multiplicative time overhead of n (i.e., simulating probabilistic time $T(n)$ in deterministic time $n \cdot T(n)$) and strong average-case derandomization with essentially no time overhead (i.e., simulating probabilistic time $T(n)$ in deterministic time $n^{\varepsilon} \cdot T(n)$ over all efficiently samplable distributions).

[13] The second assumption can be replaced by strong circuit lower bounds (see [30, Theorem 2]. Their work also deduced derandomization with $c \approx 1$ under even stronger assumptions, referring to catalytic computation (see [30, Section 1.4]).

Just as in space-bounded derandomization, there has been progress on resolving this question unconditionally, with the state of the art due to van-Melkebeek and Prakriya [81] establishing that $\mathbf{NL} \subseteq \mathbf{UL}^{3/2}$.

From the perspective of hardness-to-randomness results, we are again in the situation of assuming a large hammer to attack a possibly smaller nail (cf., Theorem 1.2). Can we do better? And again, using deterministic reconstruction, in [29, 61] they showed that the answer is yes:[14]

Theorem 3.13 ([29, 61])**.** *There is a constant $c > 1$ such that the following holds. Suppose there exists a constant $\varepsilon > 0$ such that $\mathbf{USPACE}[n]$ is hard for $\mathbf{USPACE}[cn]$-uniform circuits of size n^c with oracle access to $\mathbf{USPACE}[\varepsilon cn]$. Then, $\mathbf{NSPACE}[n] \subseteq \mathbf{USPACE}[O_\varepsilon(n)]$.*

Technically, rather than a pair $(\mathsf{GEN}, \mathsf{REC})$ with deterministic reconstruction for a distinguisher related to fooling ROBPs, we have a pair $(\mathsf{GEN}, \mathsf{REC})$ with deterministic reconstruction for a distinguisher that checks if a certain condition related to the "path-isolation lemma" from [63] holds. In particular, the distinguisher checks if a weight function $W : E \to [n^c]$ on the edges of a graph G has the property that all shortest paths are unique. The works [29, 52, 61] constructed a deterministic reconstruction for this distinguisher; see explanation in Sections 4.1 and 4.2.2.

4 Technical Tools

At a high level, in this section we will present two technical tools, both consisting of a deterministic logspace generator and a deterministic logspace (or polylogspace) reconstruction algorithm. The results presented in Section 3 rely on these tools, or on variations on these tools that will be mentioned in the text below.

Hardness of compressing a string. In Section 4.1 we describe a pair $(\mathsf{GEN}, \mathsf{REC})$ of a generator and reconstruction algorithm, which follow the explanation in Section 2. Specifically, both algorithms get access to a string f, and whenever $\mathsf{GEN}^f(D)$ fails, $\mathsf{REC}^f(D)$ compresses f to a small circuit.

Then, in Section 4.2 we expand on a key technical part in this construction (and other similar constructions), which is Distinguish to Predict transformations for restricted classes of distinguishers.

Hardness of computing a function. In Section 4.3 we describe a targeted generator and a corresponding reconstruction algorithm, which follow the explanation in Remark 2.1. Specifically, the generator will be based on a function $f(\cdot)$, and for every input x such that $\mathsf{GEN}_f(x)$ fails, the reconstruction $\mathsf{REC}_f(x)$ computes $f(x)$ using less resources than the naive algorithm for f. The construction of these algorithms will use the $(\mathsf{GEN}, \mathsf{REC})$ pair presented before that in Section 4.1.

4.1 Deterministic reconstruction via deterministic D2P

The current state-of-the-art for deterministic logspace $(\mathsf{GEN}, \mathsf{REC})$ that rely on hardness of compressing a string is the following.

[14] The precise statement below does not appear in either [29] or [61], but it can be derived by combining [61, Theorem 5.1] with [29, Theorem 4.11].

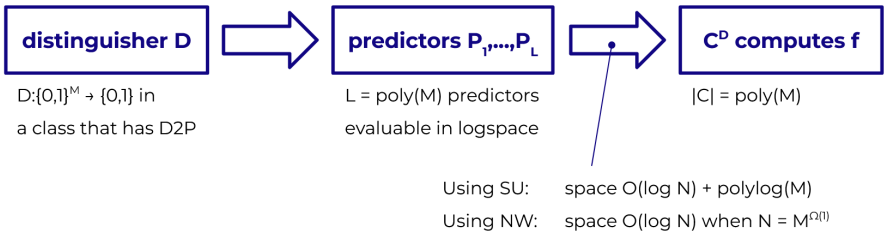


Figure 1: The high-level structure of most known reconstruction procedures, along with the parameters obtained when using the known derandomized reconstruction for the SU generator and for the NW generator.

Theorem 4.1 ([61, Theorem 5.1]; informal)**.** *Let $\mathcal{D}$ be a class of procedures such that there is an efficient distinguish-to-predict transform for $\mathcal{D}$.*[15] *There are two algorithms* (GEN, REC) *that for every $f \in \{0,1\}^N$ and every $M \leq N^{\Omega(1)}$ satisfy the following:*

- $\mathsf{GEN}(f)$ *runs in space $O(\log N)$ and outputs $\ell < \log(N)$ lists of M-bit strings.*
- *Fix any $D\colon \{0,1\}^M \to \{0,1\}$ in $\mathcal{D}$ that is a $(1/M)$-distinguisher for each of the ℓ lists that $\mathsf{GEN}(f)$ outputs. Then, $\mathsf{REC}^D(f)$ runs in deterministic space $O(\log N) + \mathrm{polylog}(M)$ and prints a circuit C of size $\mathrm{poly}(M)$ such that the truth-table of C^D is f.*

The high-level observation opening the door to Theorem 4.1, from the first paper studying deterministic reconstruction [60], is that in almost all known generator constructions (e.g., in [58, 70, 80]), the reconstruction algorithm can be divided into two steps (see Figure 1):[16]

1. Transforming the distinguisher D into a predictor P (we will define this notion in Section 4.1.2).
2. Using a predictor P, constructing a circuit C such that the truth-table of C^P is f.[17]

The two steps above often work with a list of candidate predictors, in the following sense: The first step transforms D into a list of functions $P_1, ..., P_L$, one of which is a predictor; and the second step also works given access to such a list, in which case it outputs C^{P_i} for some P_i.

In classical constructions, both steps are probabilistic. Generally speaking, the first step follows a classical easy argument of Yao [87], and most of the technical work is devoted to the second step. For example, in the classical construction of [58, 69], the second step consists of hard-wiring partial truth-tables according to a combinatorial design, and then using a decoder of an underlying locally list-decodable error-correcting code (for a standard description, see e.g. [1, Chapter 19.6]).

[15] We will define distinguish-to-predict transforms in Section 4.1.2. For Theorem 4.1, we need is the transform to be computable in deterministic logspace, and for the predictors that it outputs to be evaluable in logspace (see Remark 4.7).

[16] An exception is generators with non-deterministic reconstruction, which have been useful for studying hardness vs randomness for **AM** and for superfast derandomization (see, e.g., [23, 27, 55, 68, 71]).

[17] In Theorem 4.1 we "promised" that REC outputs C^D whose truth-table is f, whereas this step yields C^P whose truth-table is f. In known arguments P is easily evaluable given D, and hence the difference between C^D and C^P is immaterial.

It turns out that for several known generators, we can derandomize the second step in a generic way, which doesn't rely on any properties of D; this takes technical work, but does not have much conceptual innovation. We will therefore start in Section 4.1.1 by describing this more generic second step, and then in Section 4.1.2 describe the first step, which is the main interesting bottleneck.

A key difference from the time-bounded setting: Derandomizing algorithms with two-sided error. Note that in Theorem 4.1 the generator outputs ℓ lists, and (assuming that the reconstruction fails) the guarantee is only that *one of the lists is pseudorandom*. A-priori, we don't know which of the ℓ lists is the pseudorandom one, so we might take the union of the lists to yield a hitting-set generator (and derandomize algorithms with one-sided error). However, using deterministic reconstruction, we can do better.

In the special case of trying to prove that $\mathbf{BPL} = \mathbf{L}$, one trick is to rely on the fact that fooling a $\mathbf{BPL}$ machine reduces to "hitting" the set of good seeds for Nisan's PRG; in other words, derandomizing logspace algorithms with two-sided error reduces to a "hitting" problem (see Section 4.2.2 for details).

A more general approach is to observe that the *derandomization algorithm can also run the reconstruction algorithm!* (Indeed, this is because the latter is deterministic.) That is, the derandomization algorithm can compute each of the ℓ output lists of the generator, and then run the reconstruction algorithm REC on each list to see if REC succeeds; whenever REC fails, the list must be pseudorandom. See, e.g., Section 4.3.2.

Known variants of Theorem 4.1. Parameter-wise, Theorem 4.1 is not optimal (see open problems in Section 5.3). However, we know how to do better in certain special cases. Let us mention two examples:

- **The output is long:** When the output length is $M = N^{\Omega(1)}$, we can get REC that runs in space $O(\log N)$ and outputs C that is a $\mathbf{TC}^0$ circuit, rather than a general circuit. Moreover, in this case GEN also outputs a single list rather than ℓ lists. (See [28, Theorem 7.4], following [60].)

- **The distinguisher is an ROBP:** When $\mathcal{D}$ is the class of ROBPs of polynomial width, we can get REC that runs in space $O(\log N)$ and outputs C that is of size $\mathrm{polylog}(M)$, rather than $\mathrm{poly}(M)$. We will describe the proof idea in Section 4.2.2. (This result isn't explicitly stated in prior works, and it follows by combining [61, Theorem 5.1] with [29, Theorem 4.8].)

Another known alternative to Theorem 4.1 has REC that runs in near-linear time $N^{1+\varepsilon}$ and reads f in read-once fashion (i.e., bit-by-bit in order); the downside then is that this REC runs in slightly larger space $\mathrm{polylog}(N)$, and it outputs a small low-space machine that computes $j \mapsto f_j$ in time $\mathrm{poly}(|f|)$, rather than a small circuit (see [61, Theorem 2.4]). We will explain this and describe the proof idea in Section 4.3.2.

Remark 4.2. We do not know of any inherent reason to divide reconstruction arguments into the two steps above. In fact, there is evidence that the first step is an overkill,[18] and this step increases the runtime of REC at least quadratically (see [52, Appendix B]). The latter runtime overhead is an instance of what is often called "the hybrid argument barrier" (see. e.g., [32, 72]),

[18] This is since derandomizing a reconstruction argument in general is equivalent to $\mathbf{prBPP} = \mathbf{prZPP}$, whereas derandomizing the first step is equivalent to $\mathbf{prBPP} = \mathbf{prP}$; see Section 4.1.2.

and indeed the known reconstruction procedures that bypass this barrier do not involve the first step (see [23, 27], following [68]).

4.1.1 The second (generic) step

There are two known classical generators whose "second step" of reconstruction can be generically made deterministic and low-space:

- **The Shaltiel-Umans generator [70]** (with a modification from [15]): In [29, 61] they showed that for any output length $M \leq N^{\Omega(1)}$, the second reconstruction step can be implemented in deterministic space $O(\log N) + \text{polylog}(M)$. This is used to prove of Theorem 4.1.

- **The Nisan-Wigderson generator [58]** (combined with a locally list-decodable code [69]): In [60] they showed that when the output length is $M = N^{\Omega(1)}$,[19] the second reconstruction step can be implemented in deterministic space $O(\log N)$. In [28] they showed that (using a suitable code) we can in addition get the second step to output a $\mathbf{TC}^0$ circuit. This matches the special case of Theorem 4.1 with $M = N^{\Omega(1)}$ that was mentioned after the theorem's statement.

We briefly explain the proof ideas, starting with the simpler case of the NW reconstruction. The derandomization of the second step for this reconstruction replaces uniform random choices by pseudorandom choices output by an averaging sampler (i.e., a seeded randomness extractor; see [35]).

Example 4.3. One step in the reconstruction argument of the NW generator is local list-decoding of an underlying error-correcting code, which is usually chosen to be the Reed-Muller code concatenated with the Hadamard code. The decoding algorithm in [69] for the Reed-Muller code chooses a random degree-3 curve in $\mathbb{F}^m$, where the parameters are chosen such that $|\mathbb{F}|^m = \text{poly}(N)$. Choosing a curve can be done using $O(\log N)$ coins, since we just need to choose three points in $\mathbb{F}^m$ to specify the curve.

However, the basic reconstruction only succeeds with probability $1/\text{poly}(M)$, so this step is independently repeated $\text{poly}(M)$ times, resulting in high overall randomness complexity. In [60], instead of using $\text{poly}(M)$ independent choices, they use a logspace-computable sampler $\mathsf{Samp}\colon \{0,1\}^{O(\log N)} \to \left(\{0,1\}^{O(\log N)}\right)^{\text{poly}(M)}$ with accuracy $1/\text{poly}(M)$; that is, they choose coins $r \in \{0,1\}^{O(\log N)}$ for the sampler, and enumerate over the choices given by $\mathsf{Samp}(r)$ instead of over independent choices.

For the SU reconstruction, considerably more technical work went into derandomizing the second step. The idea above of replacing independent choices by samplers was used, in some cases with a special type of sampler called a "curve sampler" and a specific construction by Guo [38].

But unlike the case of the NW reconstruction, black-box replacement of independent choices by samplers is not enough here, and some parts of the reconstruction procedure needed to be

[19] This parameter setting is the typical one when using the NW generator, since this generator is not optimal when $M = N^{o(1)}$. Specifically, the NW generator has seed length $\ell = O((\log N)^2/\log(M))$, which in our terminology means that $\mathsf{GEN}^f(D)$ runs in time at least 2^ℓ. Generators with seed length $O(\log N)$ were constructed by Shaltiel and Umans [70] and by Umans [80].

refined (see [29, Section 2.1.3]). The reconstruction also does not a-priori run in low space,[20] and this required an additional idea (see [61, Section 2.2.3]). A self-contained proof appears in [61, Section 5].

4.1.2 The first step: Distinguish-to-Predict (D2P) transforms

In the previous section we established that (for some generators) the second step can be executed in deterministic logspace, and this didn't rely on any properties of D. Specifically, the procedures above just need black-box access to a predictor, so all we need now is a way to transform D into a predictor.

> The main technical bottleneck is the first step, i.e. transforming a distinguisher to a predictor. Known derandomizations of this step use the structure of D, and extending them to broader classes of D's is the key towards getting derandomized reconstruction in more settings.

We will define this precisely using a more general notion of a predictor than the standard one. Instead of allowing only "next-bit" predictors, we will also allow "previous-bit" predictors, and more generally:

Definition 4.4 (predictor)**.** For a distribution $\mathbf{w}$ over $\{0,1\}^M$, we say $P : \{0,1\}^m \to \{0,1\}$ is a δ-predictor for $\mathbf{w}$ if there is an interval $I \subseteq [M]$ where $|I| = m$ and a bit $j \in [M] \setminus I$ where

$$\mathop{\mathbb{E}}_{w\sim\mathbf{w}}[P(w_I) = w_j] \geq \frac{1}{2} + \delta.$$

We say P is a next-bit-predictor if $I = \{1, \ldots, m-1\}$, and a previous-bit-predictor if $I = \{m+1, \ldots, M\}$.

Definition 4.5 (D2P)**.** An ε-distinguish to δ-predict transform for a class C of circuits is an algorithm that takes as input an M-bit $C \in C$ and outputs a set of procedures $P_1, ..., P_L$, where each P_j is a function $\{0,1\}^i \to \{0,1\}$ for some $i \leq M$, such that the following holds. For every distribution $\mathbf{w}$ such that C is an ε-distinguisher for $\mathbf{w}$, there is $j \in [L]$ such that P_j is a δ-predictor for $\mathbf{w}$.

The requirement in Definition 4.5 that a single set $P_1, ..., P_L$ works for *all* distributions $\mathbf{w}$ is not obvious. A relaxed requirement, wherein the algorithm may take $\mathbf{w}$ as input, also makes sense (see [52, Appendix C.2]). We focus on the stronger notion, because most of the D2Ps from recent years satisfy it.

Remark 4.6. Note that a useful property of a predictor P is that, given access to $\mathbf{w}$, we can efficiently *verify* how well it predicts the relevant bit. This stands in contrast to distinguishers, since there is no obvious way to check if D is a distinguisher for $\mathbf{w}$ (since a priori we do not know the value $\mathbb{E}_{r\in\{0,1\}^M}[D(r)]$).

Remark 4.7. When using a D2P transform to construct a logspace reconstruction algorithm, we also need that the predictors given by the D2P will be evaluable in logspace. That is, we need a logspace algorithm that gets $i \in [L]$ and input w and oracle access to D, and outputs $P_i(w)$. This is important because the second step of the reconstruction (from Section 4.1) makes queries to the candidate predictors.

[20] In [29] the reconstruction is presented as a probabilistic algorithm using $O(\log N)$ coins and space $O(\log N)$. The best known way to transform this algorithm into a deterministic algorithm, from [61], pays an an extra factor of polylog(M).

D2P for general circuits. The textbook lemma by Yao [87] yields a *probabilistic* algorithm that gets a general circuit D, and for every fixed $\mathbf{w}$, whp outputs a D2P transform that works for $\mathbf{w}$.

Proposition 4.8 ([87])**.** *Suppose* $D\colon \{0,1\}^M \to \{0,1\}$ *is a* ε*-distinguisher for* $\mathbf{w}$*. Then, with probability at least* $1/\operatorname{poly}(M)$ *over choice of* $i \in [M]$ *and* $z \in \{0,1\}^{M-i}$ *and* $b \in \{0,1\}$*, the algorithm* $P : \{0,1\}^i \to \{0,1\}$ *defined as* $P(w) = D(w \circ z) \oplus b$ *is an* (ε/M)*-next-bit-predictor for* $\mathbf{w}$*.*[21]

It is not a-priori clear that for all D's there even *exists*, non-explicitly, a single small set of predictors that works for every $\mathbf{w}$ (for which D is a distinguisher). This is because there can be $2^{\Omega(2^M)}$ distributions for which D is a distinguisher,[22] so naive probabilistic arguments yield exponentially many predictors.

Nevertheless, the claim is true. The proof from [52] does not use a standard probabilistic-method argument, and instead uses a specific (non-explicit) construction of predictors.

Theorem 4.9 ([52, Theorem B.1])**.** *For every* $D : \{0,1\}^M \to \{0,1\}$ *there are* $L = O(M^2)$ *candidate predictors* $P_1, ..., P_L$ *such that the following holds. For every distribution* $\mathbf{w}$ *for which* D *is a* $(1/3)$*-distinguisher there is* $j \in [L]$ *such that* P_j^D *is an* $1/O(M)$*-predictor for* $\mathbf{w}$*.*

In a gist, the predictors in Theorem 4.9 use a (non-explicit) CAPP algorithm to estimate the expectation of certain auxiliary circuits. This is similar to an argument of Goldreich [34, Appendix A].

In particular, assuming that there is an efficient CAPP algorithm, we obtain a deterministic D2P transform for general circuits. In fact, deterministic D2P turns out to be *equivalent* to general derandomization!

Theorem 4.10 ([52, Theorem 1.5])**.** *There exists a deterministic polynomial-time* $(1/3)$*-distinguish to* $(1/\operatorname{poly}(M))$*-predict transform for general circuits* if and only if **prBPP** = **prP**.

This result is not encouraging, since it indicates that constructing D2P transforms for general circuits may be too challenging at the moment. There is even worse news: For any typical circuit class C, a non-trivial D2P transform for C implies that **NEXP** is hard for C. Specifically, an ε-distinguish to $(2^{-o(M)})$-predict transform for C that runs in time $2^M/M^{\omega(1)}$ and outputs $2^{o(M)}$ predictors (where $\varepsilon > 0$ is a small universal constant) implies non-trivial derandomization for C (see [52, Appendix B.2]). Using Williams' algorithmic method [84], such derandomization yields a problem in **NEXP** that is hard for C-circuits.

4.2 Deterministic D2P for restricted distinguisher classes

To make progress, we turn our attention to D2P for restricted classes of distinguishers, for which lower bounds in **NEXP** are either known or seem feasible to prove. We also consider D2P transforms for specific useful function classes D that don't correspond to classical circuit classes.

[21] By repeating the algorithm in Proposition 4.8 for $L = \operatorname{poly}(M)$ times, we obtain a random set of $P_1, ..., P_L$ such that for any $\mathbf{w}$, with high probability over choice of P_j's one of them is a predictor for $\mathbf{w}$. The same argument establishes that with probability $1/\operatorname{poly}(M)$ over i, z it holds that $P(w) = D(z \circ w) \oplus b$ is an (ε/M)-previous-bit-predictor for $\mathbf{w}$.

[22] To see this, consider an unbiased D, and a collection of uniform distributions over subsets of $\{0,1\}^M$ on which the expected value of D differs from $1/2$.

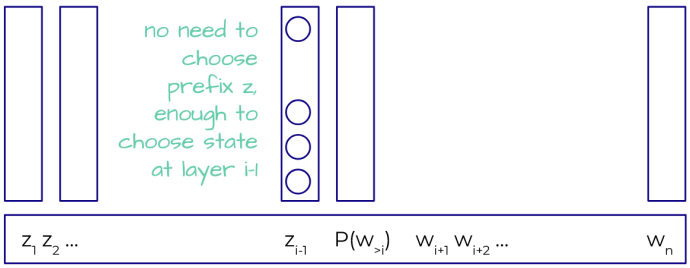


Figure 2: Visual depiction of the proof of Theorem 4.12.

4.2.1 Deterministic D2P for ROBPs

In the special case of ROBPs, there is a logspace deterministic D2P, and moreover, the latest known proof of this is simple! Let's recall the definition of ROBPs:

Definition 4.11 (ROBP)**.** An ROBP $B\colon \{0,1\}^M \to \{0,1\}$ consists of $M+1$ layers, where each layer $i \in [M]$ is labeled with a distinct input variable x_{j_i}. Each node in layer i has two outgoing edges to layer $i+1$, labeled with 0 and 1. The first layer has a "start" node s and each node in the last layer is labeled with an output value. The ROBP computes a function by starting from s and iterating through the layers $i = 1, ..., M$, reading the value v of the corresponding input bit x_{j_i}, and following the edge labeled with v to the next layer. The maximal number of nodes in any layer is the main complexity parameter, called the width.

Theorem 4.12 ([28, 60], following [11, 37, 56])**.** *There is a deterministic logspace* $(1/M)$*-distinguish to* $(1/\operatorname{poly}(M))$*-predict transform for* ROBP*s of width* W*. The algorithm outputs* $\operatorname{poly}(M, W)$ *predictors, each of which is also an* ROBP*.*

Proof. Here we finally use a special property of the distinguisher class, and a remarkably simple one. Let $D\colon \{0,1\}^M \to \{0,1\}$ be the given ROBP, and consider some (unknown) distribution $\mathbf{w}$ over $\{0,1\}^M$.

By Proposition 4.8 and Footnote 21, for some $i \in [M]$ and $z \in \{0,1\}^{i-1}$ and $b \in \{0,1\}$, the function $P_{i,z,b}(w) = D(z \circ w) \oplus b$ is a previous-bit-predictor for $\mathbf{w}$. We can enumerate over $i \in [M]$ (as we are allowed to output a list of predictors), but enumerating over all $z \in \{0,1\}^{i-1}$ is infeasible.

The key observation is that for any z, the residual function $D_z(w) = D(z \circ w)$ is a sub-ROBP of D. Specifically, after fixing the first $i-1$ bits to z, the residual function $D_z(w)$ is the ROBP obtained by starting from a node $v = v_z$ in layer $i-1$ (i.e., the node v_z we reach by walking on D according to z) and then processing the input $w \in \{0,1\}^{M-(i-1)}$. Hence, instead of enumerating over all z's, it's enough to enumerate over all v's in layer $i-1$, of which there are only at most W. See Figure 2 for a visual depiction.

The algorithm outputs the set of $2M \cdot W$ predictors obtained by enumerating all choices of $i \in [M]$ and $v \in [W]$ and $b \in \{0,1\}$, and outputting $P_{i,v,b}(w) = D_v(w) \oplus b$. □

Remark 4.13. The only property of ROBPs that this result exploits is that fixing a prefix of the input to a given ROBP D yields one of at most W sub-ROBPs of D.

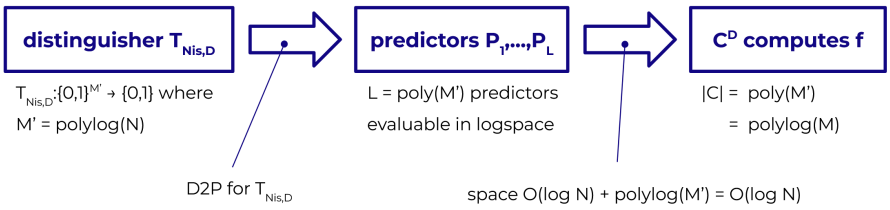


Figure 3: The high-level structure of the reconstruction procedure when using the generator Nis ∘ GEN.

Having a D2P for ROBPs is already useful, since **BPL** = **L** reduces to estimating the expected value of a given ROBP (in logspace). Using Theorem 4.12 and the derandomized logspace algorithms mentioned in Section 4.1.1, we can get a simpler version of Theorem 4.1 in which C is of size $\text{poly}(M, \log N)$ rather than $\text{polylog}(MN)$. This was used in the proofs of Theorems 3.1 and 3.3.

4.2.2 Deterministic D2P beyond ROBPs

After the statement of Theorem 4.1, we mentioned that in the special case of ROBPs we can get an improved reconstruction algorithm, which runs in space $O(\log N)$ and outputs a circuit C of size $\text{polylog}(M)$ (rather than $\text{poly}(M)$). Let us explain the idea for doing so, which will naturally lead us to consider D2P transforms for distinguishers beyond ROBPs.

Composing the generator with Nisan's PRG. Recall that we want a generator that outputs M bits and fools ROBPs. We will instantiate the generator GEN with output length $M' = \text{polylog}(M)$, and then compose it with an *unconditional PRG* that stretches M' bits to M bits and fools ROBPs. Specifically, for the "outer" PRG we will use the classical generator of Nisan [56]. That is, the final generator is Nis ∘ GEN.

The point of doing this is that when this composed generator fails (which means that GEN failed, because Nis works unconditionally), the reconstruction REC outputs C of size $\text{poly}(M') = \text{polylog}(M)$, and runs in space $O(\log N) + \text{polylog}(M') = O(\log N)$. The disadvantage, however, is that the distinguisher for the inner generator now isn't the ROBP D, but rather *Nisan's PRG composed with* D; that is, we need GEN to fool the distinguisher $D'(w) = \text{Nis}(D(w))$. This is not an ROBP anymore, and hence the crucial bottleneck in the construction of (GEN, REC) – the D2P algorithm from Theorem 4.12 – does not work anymore.

Fortunately, this can be handled! The first observation is that, inspecting the construction of Nis, for derandomization it suffices to find a good key for a pairwise-independent hash function, which is of description length $M' = \text{polylog}(N)$. Moreover, the function $T_{\text{Nis},D}$ that gets input $k \in \{0,1\}^{M'}$ and decides whether or not k is a good key for D can be implemented in logspace (this is essentially the argument in [57]; for an explanation, see [29, Section 4.1]). Thus, the distinguisher for GEN is now $T_{\text{Nis},D}$.

Theorem 4.14 ([29])**.** *There is a deterministic logspace* $(1/2)$*-distinguish to* $(1/\text{poly}(M'))$*-predict transform for the class of* $T_{\text{Nis},D}$*'s obtained from* ROBP*s* $D\colon \{0,1\}^M \to \{0,1\}$ *of width* $\text{poly}(M)$*. The algorithm outputs* $\text{poly}(M')$ *predictors, each of which is computable in logspace.*

Proof idea. Recall that Theorem 4.10 reduces D2P to **prBPP** = **prP**. Its proof shows something more specific: For any fixed $D\colon \{0,1\}^M \to \{0,1\}$, the proof gives a set of efficient predictors

$P_1, ..., P_L$ that work when given oracle access to a function that gets $z \in \{0,1\}^{i<M}$ and estimates $\mathbb{E}_{z\in\{0,1\}^{M-i}}[D(y \circ z)]$. In other words, to predict any distribution $\mathbf{w}$ (that D distinguishes from uniform), it suffices to estimate the acceptance probability of D when fixing a prefix of its input bits to a given y. For general circuits, this "prefix-CAPP" problem is equivalent to CAPP and hence complete for **prBPP**; however, in the current special we can use a trick to solve it.

Let us go back to $T_{\mathsf{Nis},D}$, which takes $k \in \{0,1\}^{O(\log N)^2}$ and decides if it is good for D. In Nisan's proof, k is partitioned into $\ell = \log(N)$ blocks of size $O(\log N)$, and k is good if and only if each block is good (for some notion of "good block" that we intentionally gloss over here).[23] Moreover, given a prefix of $i \leq \ell$ blocks in k, we can test in logspace whether all the i blocks are good.

Now let us look at the task we need to solve in order to construct a predictor. We are given a prefix y for a potential key k, and for simplicity let us assume that this is a prefix of blocks (i.e., the prefix not does end mid-block). We need to decide $\mathbb{E}_z[T_{\mathsf{Nis},D}(y \circ z)]$. The key observation is that there are only two cases:

- If there is a block in the given prefix that is not good, then $k = y \circ z$ will not be good, no matter the suffix z. Hence $\mathbb{E}_z[T_{\mathsf{Nis},D}(y \circ z)] = 0$.
- If all blocks are good, then the proof in [57] shows that a random suffix z yields a good $k = y \circ z$, with very high probability. Hence, $\mathbb{E}_z[T_{\mathsf{Nis},D}(y \circ z)] \approx 1$.

Thus, to estimate $\mathbb{E}_z[T_{\mathsf{Nis},D}(y \circ z)] = 0$ it suffices to check if all blocks in the given y are good, which we can indeed do in deterministic logspace. The D2P outputs the set of predictors from the proof of Theorem 4.10, with the required estimation task performed as above. □

To recap, the proof of Theorem 4.14 relied on a reduction from the proof of Theorem 4.10, and the main property used in the proof of Theorem 4.14 was a "polarization" effect: for any prefix y, either the acceptance probability over a random z is zero or it is very high. Moreover, we can efficiently distinguish between the two cases. Many known D2P transforms rely on similar "polarization" effects.

Fooling any-order branching programs by composing with the Forbes-Kelley PRG. For derandomization with minimal memory footprint (as in Theorem 3.11), we'd like both GEN and REC to incur almost no space overhead. In [30] they showed how to get GEN that adds essentially no space overhead, but this relies on one modification: instead of considering a distinguisher D that is an ROBP, we now need to consider D that is an Any Order Branching-Program (AOBP). For a definition see, e.g., [16, 31], though knowing the precise definition is not crucial when reading the current text. Unfortunately, the Nisan generator that we constructed a D2P transform for provably does *not* fool this model [79]!

To get a hyper-efficient GEN along with REC that deterministically compresses to size polylog(M), the idea in [28] was to replace Nisan's PRG with an unconditional PRG for AOBPs that can be evaluated in extremely low space, and such that there is a D2P for the composition of this PRG with an AOBP. It turns out that a modification of the Forbes-Kelley PRG [31] satisfies all these properties.

Theorem 4.15 (informal; see [29, Section 4.3])**.** *For any $\varepsilon > 0$, there is a deterministic algorithm that gets as input an* AOBP $D\colon \{0,1\}^M \to \{0,1\}$ *of size M and a distribution* $\mathbf{w}$ *over* $\{0,1\}^{M^\varepsilon}$ *for*

[23] To be more accurate, in the original proof this is not an "if and only if", since some k's happen to be good even if not all blocks are good. However, for our purposes we modify the definition of "a good k" to mean that all blocks are good.

which $D \circ FK$ is a $(1/10)$*-distinguisher,*[24] *runs in time* $\mathrm{poly}(M, |\mathbf{w}|)$ *and space* $O(M^{\varepsilon} + \log |\mathbf{w}|)$*, and outputs a* $(1/M^{O(\varepsilon)})$*-predictor P for* $\mathbf{w}$*. The predictor can be evaluated in space* $(1 + O(\varepsilon)) \cdot \log(M)$ *with access to D.*

Observe that Theorem 4.15 achieves the more relaxed notion of D2P that was mentioned after Definition 4.5. At a high-level, the proof of Theorem 4.15 uses a "polarization" idea similar to the proof of Theorem 4.14, but significantly more technical work is needed to carry it through, along with modifications to the original FK generator. See [29, Sections 2.3.1 and 4.3] for a proof description. The resulting (GEN, REC) pair is implicit in the proof presented in [29, Section 6.4].

Path isolation by composing with the van Melkebeek and Prakriya PRG. There is also a deterministic D2P for a completely different type of distinguisher, which doesn't arise from ROBPs/AOBPs.

Recall that in Section 3.4.3 we mentioned a result from [29, 52, 61] deducing $\mathbf{NSPACE}[n] \subseteq \mathbf{USPACE}[O(n)]$ from lower bounds for $\mathbf{USPACE}[O(n)]$-uniform circuits. As mentioned there, that result is based on a generator that produces weight assignments for the edges a given graph, where the distinguisher $D(W)$ checks whether or not the weight assignment W induces unique shortest paths in the graph. Getting deterministic $\mathbf{USPACE}[O(n)]$-reconstruction for this generator calls for designing a deterministic D2P for D.

Reinhardt and Allender [63] showed that D can be implemented in unambiguous logspace (i.e., in $\mathbf{UL}$, which in the parameter above gives a $\mathbf{USPACE}[O(n)]$ algorithm), but their algorithm does not seem at all like an ROBP/AOBP. Fortunately, in [52] they still showed a D2P for it. Their idea was to mimic the proof of Theorem 4.14: They reduced D2P to solving a "prefix-CAPP" problem for D, observed that to solve the latter it suffices to check whether the given prefix violates a condition related to D, and relied on the algorithm of [63] for D to decide the latter. Details appear in [52, Section 2.2].

4.3 New reconstructive targeted generators for the logspace setting

We now design a generator and reconstruction based on hardness of *computing a function* $f(x)$, rather than of compressing a fixed string f. That is, we fix a uniformly computable function $x \mapsto f(x)$, and both the generator and the reconstruction will get input x. Intuitively, the generator tries uses the computation of $f(x)$ as a source of hardness (which it can convert to pseudorandomness).

The technical tool that we will rely on is a targeted PRG, as introduced by Goldreich [34,36]. This is a generator that gets input x and tries to fool efficient uniform algorithms that also receive the same input x (i.e., rather than trying to fool all small non-uniform circuits, as in classical PRGs). The reconstruction algorithm also gets x, and if the generator fails, then the reconstruction manages to compute $x \mapsto f(x)$ "too efficiently". We stress that the analysis is now performed instance-wise, for every fixed input x. That is, if $x \mapsto f(x)$ is hard to compute on x specifically, then the generator with x produces a distribution that is pseudorandom for uniform algorithms that get access to this specific x.

We also note that the reconstruction in this setting does not get oracle access to a hard string anymore (let alone oracle access to f). The reconstruction simply gets x and tries to compute $f(x)$.

[24] The version of the Forbes-Kelley PRG from [29] stretches M^{ε} bits to M bits.

A concrete targeted generator. The generator below, from [61] (following [29]), is based on the supposed hardness of composing low-space algorithms "for free", i.e. without a significant overhead either in the running time or in the space complexity (see Section 3.3.2 for an exposition of this topic). Specifically, fix an arbitrary f_0 computable in logspace and large polynomial time T. The result below gives a pair of algorithms that, on every input, either compute the k-fold composition of f_0 in *near-linear* time $T^{1+\varepsilon}$ and polylogarithmic space, or derandomize **BPL** in space $O(k \cdot \log(n))$.[25]

Theorem 4.16 (win-win pair of algorithms for derandomization and composition; [61], following [29])**.** *Let $L \in$ **BPL**, let $\varepsilon > 0$, and let T be a sufficiently large polynomial. Fix a length-preserving f_0 computable in logspace and linear time, and $k = k(n)$. Then, there are two algorithms A, B such that for every $x \in \{0,1\}^n$, at least one of the following holds:*

- *$A(\bar{x}) = f_0^{(k)}(\bar{x})$ in time $T^{1+\varepsilon}$ and space* polylog(n)*, where $\bar{x} = x0^{T-n}$.*
- *$B(x) = L(x)$ in space $O(k \cdot \log(n))$.*

In terms of hardness vs randomness, Theorem 4.16 can be parsed as follows: if computing k-wise composition is infeasible in near-linear time and low space, then we can derandomize **BPL**. We will explain below how the generator's construction can yield not only Theorem 3.8 but also Theorem 3.5.

For notational convenience, in the rest of the section we write $f(x) = f_0(x)$, and note that the hard function on which the generator is based is $x \mapsto f^{(k)}(x)$.

4.3.1 A bootstrapping system for low-space algorithms

The main technical challenge is that the reconstruction algorithm now gets input x and does not have query access to $f^{(k)}(x)$. In particular, it is not a-priori clear how to utilize classical generators, whose reconstruction algorithms need query access to $f^{(k)}(x)$. The construction underlying Theorem 4.16, which handles this challenge, is an adaptation of the generator by Chen and Tell [22] (using ideas from [47]).

The generator is given input $x \in \{0,1\}^n$, and for simplicity let us abuse the notation and use x to also denote the padded T-bit version $x0^{T-n}$. Consider the sequence of strings $f^{(1)}(x) = f(x)$, $f^{(2)}(x) = f(f(x))$, ... and so on until $f^{(k)}(x)$. Note that this sequence of strings is "downward self-reducible", in the sense that $f^{(i)}(x)$ is computable in logspace from $f^{(i-1)}(x)$. See Figure 4.

The targeted generator applies the generator GEN from Theorem 4.1 to each of the strings $f^{(i)}(x)$, and obtains a sequence of lists $L^{(i)} = \mathsf{GEN}(f^{(i)}(x))$.[26] The hope is that at least one of those lists will be pseudorandom for the **BPL** machine with the same input x. Indeed, assume that this is not the case. Then, the deterministic reconstruction algorithm REC from Theorem 4.1 runs in logspace and builds a small circuit C_1 (of size polylog$(M) \leq$ polylog(T)) whose truth-table is $f^{(1)}(x)$, as long as REC can make queries to the bits of $f^{(1)}(x)$. The key point is that we can compute answers to these queries in logspace, due to downward self-reducibility and since we have x; that is, when REC queries $f^{(1)}(x)$ at location i, we compute the answer by running the logspace algorithm that computes $f^{(1)}(x) = f(x)$ from x. Continuing this further, at each iteration i we have a circuit $C^{(i)}$ of size polylog(T) whose truth-table is $f^{(i)}(x)$, and REC builds a small

[25] To capture a more general setting, the result models f as a linear-time function mapping T bits to T bits, and pads the input x to be of length T. Other models work just as well, e.g. considering an initial function mapping n bits to T bits and then $k-1$ length-preserving functions.

[26] To get the statement of Theorem 4.16 specifically, we assume that T is a large enough polynomial so that the output length $M = T^{\Omega(1)}$ suffices to derandomize the **BPL** machine for L.

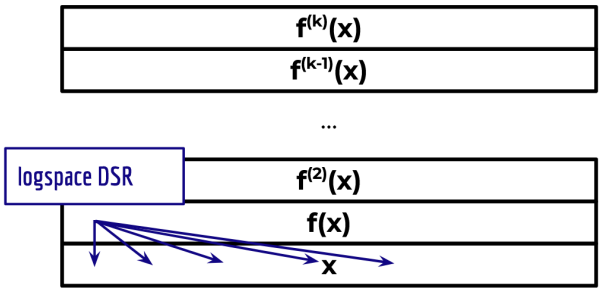


Figure 4: A boostrapping system based on repeated composition of a logspace-computable function.

circuit $C^{(i+1)}$ whose truth-table is $f^{(i+1)}(x)$, using $C^{(i)}$ and downward self-reducibility to answer queries to $f^{(i+1)}(x)$. (And after the iteration $C^{(i)}$ is discarded.) Note that we use the assumption that the **BPL** machine with x is a distinguisher for each list $L^{(i)}$ (since we want REC to succeed on each $f^{(i)}(x)$).

After k iterations the reconstruction algorithm obtained a small circuit $C^{(k)}$ whose truth-table contains the output $f^{(k)}(x)$, so it can evaluate this circuit to compute $f^{(k)}(x)$. This reconstruction runs in time poly(T), rather than $T^{1+\varepsilon}$ as promised in Theorem 4.16; in Section 4.3.2 we will explain how to improve the running time. But the space complexity of the reconstruction is polylogarithmic as we wanted, so as long as $f^{(k)}(x)$ is hard to compute in time poly(T) and space polylog(T), one of the output lists of the targeted generator fools the **BPL** machine with x.

Remark 4.17. The targeted generator above outputs k lists $L^{(1)}, ..., L^{(k)}$, hoping that at least one of them will be pseudorandom. In fact, inspecting things more closely, each $L^{(i)}$ is, in itself, a sequence of $\ell < \log(T)$ lists (because $L^{(i)} = \mathsf{GEN}(f^{(i)}(x))$, and GEN outputs ℓ lists), and if the reconstruction fails, then only one list inside one $L^{(i)}$ is guaranteed to be pseudorandom. In other words, the construction above is of a targeted "somewhere-PRG", rather than a targeted PRG. Fortunately, as explained in the beginning of Section 4.1, in the logspace setting we can often still use such an object to derandomize algorithms with two-sided error.

A win-win pair of algorithms for *s*-*t* connectivity. The construction above also suffices to prove Theorem 3.5, using one additional observation. Specifically, in this case the function f will be matrix squaring, and the generator uses $k = O(\log n)$ compositions to compute the transitive closure of the graph. The proof described above works as-is, and the only problem is that the generator's complexity is now $O(k \cdot \log(n)) = O(\log^2(n))$ (which is not useful for derandomization of **BPL**, as derandomization in space $O(\log^{3/2}(n))$ is already known unconditionally [74]).

The observation in [29] is that in this special case, we can compute each entry of $f^{(i)}(x)$ (for any $i \in [k]$) in **NL**, since computing such an entry reduces to a connectivity question in a graph. Hence, the generator is computable in non-deterministic logspace rather than only deterministic space $O(k \cdot \log(n))$, and the win-win pair of algorithms either solves *s*-*t* connectivity or derandomizes **BPL** in **NL**.

4.3.2 Faster reconstruction: The Tree-PRG

The construction described in Section 4.3.1 is essentially from [29], and the missing piece in the proof of Theorem 4.16 is to get reconstruction in time $T^{1+\varepsilon}$ rather than poly(T). A targeted generator with such reconstruction was shown in [61], using an additional construction on top of the one from Section 4.3.1.

Where is the problem, and what do we need? To explain the idea, observe that the poly(T) time overhead in reconstruction comes from two sources, both of them occurring when running the reconstruction algorithm REC from Theorem 4.1 (i.e., that compresses $f^{(i+1)}(x)$ at each layer $i+1$):

1. The running time of REC is a large polynomial poly(T).
2. Whenever REC tries to read a bit of its input $f^{(i+1)}(x)$, we run the downward self-reducibility algorithm DSR (answering its queries using $C^{(i)}$) and discard all but the relevant output bit. That is, we compute $\mathsf{REC}(\mathsf{DSR}(C^{(i)}))$ using emulative composition, incurring a quadratic time overhead (at least).

To handle the first problem, we need reconstruction that runs in near-linear time $T^{1+\varepsilon}$ instead of time poly(T). To handle the second problem, the observation in [61] is that if REC would have been read-once (i.e., if REC would read its input $f^{(i+1)}(x)$ bit-by-bit, in order, making only a single pass overall), then this problem would disappear. This is because we could run REC, and whenever it needs the next bit of $f^{(i+1)}(x) = \mathsf{DSR}(C^{(i)})$, we would run DSR from its current state until it produces the next bit, then "freeze" the execution of DSR and store its state until REC needs the subsequent bit.

A $(\mathsf{GEN}, \mathsf{REC})$ with reconstruction satisfying both conditions was shown in [61]. Specifically, they showed the following incomparable alternative to Theorem 4.1, which was mentioned after its statement:

Theorem 4.18 ([61, Theorem 2.4])**.** *There are algorithms* $(\overline{\mathsf{GEN}}, \overline{\mathsf{REC}})$ *such that for every* $\varepsilon > 0$ *and every* $f \in \{0,1\}^N$ *and* $M \leq N^{\Omega_\varepsilon(1)}$ *satisfy the following:*

1. $\overline{\mathsf{GEN}}(f)$ *runs in space* $O(\log N)$ *and outputs* $\ell < \log(N)$ *lists of* M*-bit strings.*
2. *Fix any* ROBP $D\colon \{0,1\}^M \to \{0,1\}$ *that is a* $(1/M)$*-distinguisher for each of the* ℓ *lists that* $\overline{\mathsf{GEN}}(f)$ *outputs. Then,* $\overline{\mathsf{REC}}^D(f)$ *runs in time* $N^{1+\varepsilon}$ *and space* polylog(N)*, reads* f *in read-once fashion, and prints a* polylog(N)*-size oracle machine* A *such that* A^D *describes* f*. (Specifically, given* $j \in [N]$*, the machine* $A^D(j)$ *runs in space* polylog(N) *and time* poly(N) *and outputs* f_j*.)*

We will be using Theorem 4.18 with $N = T = |f^{(i)}(x)|$, and stated it with the notation N to facilitate a comparison with Theorem 4.1. Indeed, the two improvements over the latter are that $\overline{\mathsf{REC}}$ is faster and read-once, as we needed. The price we pay is that $\overline{\mathsf{REC}}$ uses space polylog(N), rather than $O(\log N)$; and that the compressed representation of f is not a small circuit, but rather a small oracle machine whose running time is larger than $|f|$.

Read-once reconstruction: A generic transformation. In [61] they showed a generic transformation of reconstructive generators into reconstructive generators whose reconstruction is

read-once. Specifically, starting from $(\mathsf{GEN}, \mathsf{REC})$, consider the generator whose output lists consist of

$$\mathsf{GEN}(f_{\leq 1}), \quad \mathsf{GEN}(f_{\leq 2}), \quad \mathsf{GEN}(f_{\leq 3}), \quad ..., \mathsf{GEN}(f_{\leq i}), \quad ..., \mathsf{GEN}(f),$$

where $f_{\leq i}$ is the i-bit prefix of f (padded to length N). In words, the new generator applies GEN to each prefix of f, and its output collection of lists is the union of the lists output by GEN for all prefixes.

How does this yield a read-once reconstruction? The new reconstruction algorithm works iteratively. For $i = 1, ..., N$, it stores a small circuit whose truth-table is $f_{\leq i}$ (the base case $i = 1$ is trivial). Then it reads the next bit f_{i+1}, at which point it has all the information needed to answer queries to $f_{\leq i+1}$. It thus runs $\mathsf{REC}(f_{\leq i+1})$ to obtain a small circuit for $f_{\leq i+1}$, and continues to the next iteration. See [61, Section 6.1].

Remark 4.19. Since the new reconstruction needs to evaluate the stored circuit at each iteration, it now uses space $\text{polylog}(M) \leq \text{polylog}(N)$ rather than only $O(\log N)$. This seemingly matches the parameters in Theorem 4.18, but since later on we will construct a generator that needs to compute the output of this read-once reconstruction (and we want the generator to run in space $O(\log N)$), the current space complexity is not good enough. For simplicity, let us pretend that so far the reconstruction uses space $O(\log N)$.[27]

Near-linear time reconstruction: Another generic transformation. Let us assume from now on that $(\mathsf{GEN}, \mathsf{REC})$ are such that REC is read-once, but runs in large polynomial time. Recall that these $(\mathsf{GEN}, \mathsf{REC})$ originate from the Shaltiel-Umans PRG [71], so an obvious attempt would be to directly optimize the latter's reconstruction. However, this strikes us as challenging to solve directly, since this reconstruction algorithm is involved and repeatedly composes "heavy" pseudorandom primitives.

Instead, to handle this large runtime (i.e., the problem in Item 1 above), the idea in [61] is to never run GEN or REC on strings of length N – only on shorter strings, of length, say, M. In this case the runtime overhead of REC is $\text{poly}(M) \leq N^{1+\varepsilon}$, where the inequality holds if N is a large enough polynomial in M.

The construction of $\overline{\mathsf{GEN}}$ is visually depicted in Figure 5. In words, the generator $\overline{\mathsf{GEN}}$ splits f into chunks of length M, and applies GEN to each chunk to obtain a sequence of output lists. It then *runs the original reconstruction* REC *on each chunk* $j \in [N/M]$, to obtain a circuit $C^{(j)}$ of size $\text{polylog}(M)$ whose truth-table is, supposedly, the j^{th} chunk (indeed, REC may fail to output a valid circuit, but we can treat its output as a string of the relevant length nonetheless). Note that this crucially uses the fact that REC is a deterministic logspace algorithm (because we want $\overline{\mathsf{GEN}}$ to also be a deterministic low-space algorithm). Concatenating the outputs of REC on all N/M chunks, we obtain a new string $f^{(1)} \in [(N/M) \cdot \text{polylog}(M)]$. Now $\overline{\mathsf{GEN}}$ recurses, splitting $f^{(1)}$ into chunks of length M, applying GEN to each chunk to obtain a sequence of output lists, and using REC to compress $f^{(1)}$ to $f^{(2)}$ of length $\tilde{O}(N/M^2)$, and so on.

After constantly many iterations (so that the final step is applied to one remaining M-bit chunk), the generator $\overline{\mathsf{GEN}}$ accumulated output lists of GEN from chunks in each layer, and it outputs the set of all these lists. Observe that the new generator can be computed in space

[27] In [61] they handle this problem by showing two reconstruction algorithms, one that is read-once and runs in space $\text{polylog}(N)$ and time $\text{poly}(N)$, and another that is not read-once and runs in space $O(\log N)$, such that both algorithms produce the same output. The generator will use the latter.

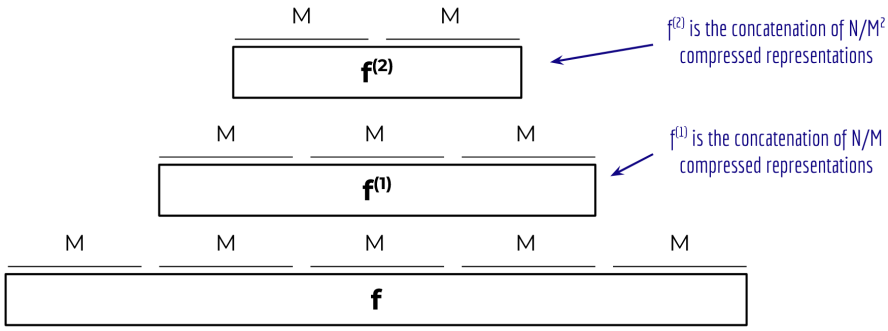


Figure 5: Visual depiction of the tree-PRG.

$O(\log N)$, using emulative composition (i.e., the standard DFS-style simulation of circuits) to compute each chunk in each layer.[28]

Indeed, it is instructive to imagine this construction as a constant-depth tree, wherein each node is labeled by a polylog(M)-size circuit and has fan-in M. The reconstruction $\overline{\mathsf{REC}}$ will output a machine that has the label of the tree's root hard-wired, and that – when given index $i \in [N]$ – computes the labels of the nodes along path in the tree, and finally outputs the i^{th} bit of f. Computing the labels along the path is done by repeatedly evaluating each label; that is, each label is a polylog(M)-sized circuit whose truth-table is the next M-bit chunk, which contains the next label.

The only missing piece is explaining how $\overline{\mathsf{REC}}$ computes the label of the root (to hard-wire it into the machine above). At a high-level, it does so using emulative composition (i.e., the DFS-style algorithm, similarly to the generator), but to prevent a polynomial runtime overhead, it crucially uses the fact that REC at each layer is read-once. Specifically, $\overline{\mathsf{REC}}$ runs REC on the M-bit chunk that yields the root's label, providing REC virtual access to this chunk; the latter virtual access is provided by running REC and providing it virtual access to the chunks below it, and so on. The crucial point is that at each level, REC reads the relevant chunk bit-by-bit, in order; hence, at any given moment $\overline{\mathsf{REC}}$ only needs one path in the tree, and each path is only used once. Using the idea of computing the composition of a read-once f with a low-space g in near-linear time, by continuously storing the state of g (as described after Theorem 4.18), this procedure can be implemented in space polylog(N) and time $N \cdot \text{poly}(M) = N^{1+\varepsilon}$. See [61, Section 6.2] for details.

5 Open Problems

The big open problem that recent results suggest is to design more low-space algorithms using hardness vs randomness. In this section we suggest several interesting algorithmic problems that may be tackled using hardness vs randomness, as well as open problems related to improving the technical machinery underlying hardness vs randomness in the small-space setting.

[28] This statement assumes, for simplicity, that the reconstruction (which the generator needs to compute) runs in space $O(\log N)$. As stated in Remark 4.19, in [61] they show a second reconstruction algorithm which produces the same output as the one described above, and runs in space $O(\log N)$ (but is not read-once, which is a feature the generator does not need).

5.1 Algorithms using hardness vs randomness

Of course, the grandparent of the open algorithmic problems in this setting is to actually resolve that pesky 5-decades old question of derandomizing logspace. To make things (potentially) easier, we also raise the question of an average-case or scaled-up result:

Problem 1. Prove that $\mathbf{BPL} = \mathbf{L}$ (or $\mathbf{BPL} \subseteq \mathbf{avgL}$ or $\mathbf{BPSPACE}[O(n)] = \mathbf{SPACE}[O(n)]$) using hardness vs randomness; that is, reduce the problem to lower bounds that we can unconditionally prove.

As a more modest step along this direction:

Problem 2. Reduce $\mathbf{BPL} = \mathbf{L}$ (or $\mathbf{BPL} \subseteq \mathbf{avgL}$ or $\mathbf{BPSPACE}[O(n)] = \mathbf{SPACE}[O(n)]$) to lower bounds that are weaker than the ones in Theorems 3.9 and 3.10.

As explained in Section 3.2, a main technique for building catalytic algorithms is "compress or random", in which the catalytic tape is either useful or can be compressed. Since deterministic reconstruction provides a generic way to compress strings that are not good (i.e., that are not useful for fooling certain distinguishers), it is an obvious tool for generalizing "compress or random" to broader settings.

Problem 3. Design **catalytic algorithms** for interesting problems, in particular general problems (e.g., simulating classes of algorithms), using "compress or random" and deterministic reconstruction.

Hardness vs randomness tools were recently used by Chen *et al.* [15] to construct *pseudodeterministic* algorithms that (unconditionally) solve a broad class of explicit construction problems. The reason their algorithm is only pseudodeterminsitic, rather than deterministic, is that the reconstruction algorithm that they use is randomized. Given that in the logspace setting we have deterministic reconstruction algorithms, a natural direction is to try and use the same approach to design unconditional *deterministic* algorithms for explicit construction problems in the logspace setting.

Problem 4. Design new *deterministic* algorithms for interesting **explicit construction problems**, by leveraging hardness vs randomness with deterministic reconstruction.

Another unconditional algorithm using hardness vs randomness was shown by Carmosino *et al.* [12]. Loosely speaking, they instantiated a PRG that is intentionally faulty and can be broken, and deduced that the reconstruction unconditionally works. In their setting the reconstruction is a probabilistic learning algorithm, and indeed, they obtained a probabilistic learning algorithm for $\mathbf{AC}^0[\oplus]$. A natural direction is to try and use the same approach with deterministic reconstruction (which can be cast as a learning algorithm, cf. Theorem 4.1) to design unconditional deterministic learning algorithms for interesting classes.

Problem 5. Obtain **deterministic learning algorithms** for interesting function classes, by leveraging hardness vs randomness with deterministic reconstruction.

The assumption that composing low-space algorithms requires significant overhead in time or in space (i.e., the hypothesis in Theorem 3.8) *looks* natural, but should we believe that it is true? Remarkably, there is very little complexity-theoretic evidence that this assumption is true (as well as the corresponding assumption for k-wise composition, which was mentioned after Theorem 3.8). We believe that it is an important problem to establish complexity-theoretical foundations for this assumption.

Problem 6. Show that lower bounds on **composing low-space algorithms** (as in Theorem 3.8, or similar lower bounds) follow from natural and/or well-studied assumptions.

5.2 Distinguish-to-predict

As explained in Section 4.2, the key to extending deterministic reconstruction to broader settings is designing distinguish-to-predict transformations for more distinguishers.

Recent works constructed D2P transformations for ROBPs [28, 60], for ROBPs composed with certain PRGs for logspace [29], and for tests related to the isolation lemma [52]. As demonstrated by the D2P for the path-isolation lemma, it can be useful to consider specific types of distinguishers, which do not necessarily conform to classical complexity classes. We pose designing deterministic D2P for new types of distinguishers as an important and general open problem.

Problem 7. Construct a D2P transform for new types of distinguishers, and obtain (using the already available tools) generators with deterministic reconstruction for such distinguishers.

In the opposite direction, recall that for general circuits D, a D2P transform for D yields derandomization of D, i.e. an algorithm estimating $\mathbb{E}[D]$ (see Theorem 4.10). This result was not an obstacle for the works surveyed in this text, since constructing a D2P transform only yields a polynomial-time algorithm for estimating $\mathbb{E}[D]$ rather than a logspace algorithm, even if the D2P transform runs in logspace. The bottleneck in the proof is a construction of an unpredictable distribution by Goldreich and Wigderson [41], which uses large space (and is highly sequential).

Problem 8. Show a reduction of estimating $\mathbb{E}[D]$ for a given circuit D to constructing a D2P transform for D such that the reduction incurs as little complexity overhead as possible.

Another interesting open problem is to expand the reach of D2P by allowing more general notions of "predictor". Indeed, most predictors considered in the literature are next-bit-predictors, but list-predictors over large alphabets have been used extensively in the past (as one example see, e.g., [78]), and replacing next-bit-predictors with previous-bit-predictors has been useful for constructing D2P for ROBPs.

Problem 9. Is it easier to construct D2P transforms if we allow more general notions of predictors that can still be useful, such as list-predictors over non-binary alphabets (with a small list) or predict-one-out-of-k-symbols (for a small k)?

A specific interesting notion of a bit-predictor allows the predictor access to *all* bits except the one it tries to predict (cf., Definition 4.4). Interestingly, the polynomial-time algorithm of [41] that constructs an unpredictable distribution (for a given collection of predictors) does not work as-is with this more general definition of predictors (see Theorem A.4). There is a natural **ZPP** algorithm for this problem (guess a random distribution and verify all predictors do not achieve good advantage). Can we construct a *deterministic* polynomial-time algorithm, or would such an algorithm imply some breakthrough?

Problem 10. Give a polytime algorithm (or evidence that such an algorithm would imply new results) for the following problem: Given a collection of poly(n) predictors $P_i : \{0,1\}^{n-1} \to \{0,1\}$, each of which attempts to predict a bit j_i, output a distribution $\mathcal{S}$ that is $(1/n)$-unpredictable to all of them.

Lastly, perhaps the most important open problem in this context is to design reconstruction algorithms *without* D2P. As explained in Remark 4.2, relying on D2P in the reconstruction causes an inherent runtime overhead (which is an instance of the hybrid argument barrier), and there is concrete evidence that D2P is an overkill for reconstruction (since deterministic

reconstruction is equivalent to **prZPP** = **prBPP**, whereas deterministic D2P is equivalent to the stronger **prBPP** = **P**). Remarkably, known reconstruction algorithms *all* fall into two camps: algorithms that use D2P, or algorithms that use non-determinism (e.g., as in [23, 27, 55, 68, 71]). Can we design a reconstruction algorithm that avoids both pitfalls?

Problem 11. Build a reconstructive pseudorandom generator whose reconstruction does not rely on nondeterminism *or* on a D2P transformation.

5.3 Generators for the logspace setting

In addition, we call for improving the main part of the hardness vs randomness framework in the logspace setting: designing better pseudorandom generators and reconstruction arguments. There are several directions that strike us as tractable and that are likely to have applications.

First, consider the targeted generator in Theorem 4.16, which is based on hardness of composing low-space algorithms. Can we design a targeted generator based on other types of hard functions, ideally more general function classes or more relaxed types of hardness?

Problem 12. Design a targeted generator that works in logspace and whose reconstruction works in deterministic logspace (or polylogspace), based on additional types of hard functions, beyond k-wise compositions of low-space algorithms for a small k.

A core technical component in the targeted generator constructions is the $(\overline{\mathsf{GEN}}, \overline{\mathsf{REC}})$ pair of Theorem 4.18, or alternatively the $(\mathsf{GEN}, \mathsf{REC})$ pair of Theorem 4.1, both of which are based on hardness of compressing a string. We believe that these two tools can be improved in a few ways.

First, the space complexity of the two reconstruction procedures is sub-optimal, i.e. $\text{polylog}(N)$ and $O(\log N) + \text{polylog}(M)$, respectively. This is because these two results rely on the SU PRG, and their reconstruction uses a derandomized version of the SU reconstruction with space complexity is $O(\log N) + \text{polylog}(M)$.[29] Given that [28, 60] showed a deterministic reconstruction for the NW PRG using space $O(\log N)$, it seems plausible that the reconstruction for the SU PRG can also be made to run in such space. (This is important because the NW PRG is suitable mostly when the output length is $M = N^{\Omega(1)}$, and the SU PRG is suitable more generally for any $M \leq N^{\Omega(1)}$; see Footnote 19.)

Problem 13. Improve the reconstruction algorithm for the SU generator so that it runs in space $O(\log N)$ when given access to a predictor. Alternatively, present another $(\mathsf{GEN}, \mathsf{REC})$ pair achieving the parameters as Theorem 4.1 but with reconstruction that runs in space $O(\log N)$ when given access to a predictor.

Secondly, the generator in Theorems 4.1 and 4.18 is a somewhere-PRG, rather than a PRG. As explained in Section 4.1, this disadvantage does not affect the specific results in this survey, since for the problems considered in this survey there are ways around it. However, it would be ideal to have a pseudorandom generator with deterministic reconstruction (and optimal parameters, unlike NW), rather than only a hitting-set generator. The natural candidate is the PRG by Umans [80], which is a refinement of the SU generator and thus its reconstruction may be amenable to similar derandomization techniques.

Problem 14. Derandomize the reconstruction procedure of Umans' [80] PRG (i.e., the reconstruction parts that follow the D2P transform).

[29] In Theorem 4.18 the reconstruction incurs an additional overhead due to the tree compression procedure described in Section 4.3.2.

Another possible improvement is to have a reconstruction algorithm that compresses the hard string to an oracle circuit *from a weak circuit class*. Indeed, classical non-uniform reconstruction arguments do yield weak circuits (e.g., $\mathbf{TC}^0$ circuits, and this is true even in the logspace setting [30]), as does the deterministic reconstruction for the NW generator in [60]. However, the reconstruction arguments in Theorems 4.1 and 4.18 have no such guarantee. One concrete goal to shoot for is having a procedure from a class that is evaluable in logspace, either a circuit from a weak class or a Turing machine running in logspace.

Problem 15. Improve Theorems 4.1 and 4.18 such that the output of the reconstruction algorithm can be evaluated in space $O(\log N)$.

A related improvement, which is specific to Theorem 4.18, refers to the fact that the reconstruction in that result outputs an oracle machine A such that A^D computes f, but evaluating the machine with input j to produce f_j takes time $t \gg |f|$. This stands in contrast to standard reconstruction arguments (e.g., in [58, 71], or their deterministic counterparts in [28, 29, 60, 61]), which output a circuit C of size $|C| \ll |f|$ such that C^D computes f.

Problem 16. Improve Theorem 4.18 such that the reconstruction algorithm outputs a circuit C of size $\mathrm{poly}(M, \log(N))$.

A last open problem may have applications when considering derandomization with overhead that is simultaneously minimal in both time and space. In the results mentioned in Section 3.4.2, the generator has very low space consumption, but it may be slow, and ditto for the reconstruction. In Theorems 4.16 and 4.18 the reconstruction's running time is optimized to be essentially linear, but the generator may still be slow (i.e., GEN runs in time $\mathrm{poly}(|f|)$ for some large unspecified polynomial). Is this improvable?

Problem 17. Improve Theorem 4.18 such that GEN runs in time $|f|^{1+\varepsilon}$.

References

[1] Sanjeev Arora and Boaz Barak. *Computational complexity: A modern approach*. Cambridge University Press, Cambridge, 2009.

[2] Alexander E. Andreev, Andrea E. F. Clementi, and José D. P. Rolim. A new general derandomization method. *Journal of the ACM*, 45(1):179–213, 1998.

[3] Alexander E. Andreev, Andrea E. F. Clementi, José D. P. Rolim, and Luca Trevisan. Weak random sources, hitting sets, and BPP simulations. *SIAM J. Comput.*, 28(6):2103–2116, 1999.

[4] Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting uniform and nonuniform upper bounds. *J. Comput. Syst. Sci.*, 59(2):164–181, 1999.

[5] Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A sublinear space, polynomial time algorithm for directed *s*-*t* connectivity. *SIAM J. Comput.*, 27(5):1273–1282, 1998.

[6] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *Proc. 46 Annual ACM Symposium on Theory of Computing (STOC)*, pages 857–866, 2014.

[7] Marshall Ball, Lijie Chen, and Roei Tell. Towards free lunch derandomization from necessary assumptions (and owfs). In Srikanth Srinivasan, editor, *40th Computational Complexity Conference, CCC 2025, Toronto, Canada, August 5-8, 2025*, volume 339 of *LIPIcs*, pages 31:1–31:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[8] Harry Buhrman and Lance Fortnow. One-sided versus two-sided error in probabilistic computation. In *Proc. 16th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 100–109, 1999.

[9] László Babai, Péter Hajnal, Endre Szemerédi, and György Turán. A lower bound for read-once-only branching programs. *Journal of Computer and System Sciences*, 35(2):153–162, 1987.

[10] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[11] Kuan Cheng and William M. Hoza. Hitting sets give two-sided derandomization of small space. *Theory Comput.*, 18:1–32, 2022.

[12] Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, Tokyo, Japan, May 29 - June 1, 2016*, volume 50 of *LIPIcs*, pages 10:1–10:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[13] Lijie Chen, Jiatu Li, and Jingxun Liang. Maximum circuit lower bounds for exponential-time Arthur Merlin. In *Proc. 57th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1348–1358, [2025] ©2025.

[14] James Cook, Jiatu Li, Ian Mertz, and Edward Pyne. The structure of catalytic space: Capturing randomness and time via compression. *Electron. Colloquium Comput. Complex.*, TR24-106, 2024.

[15] Lijie Chen, Zhenjian Lu, Igor Carboni Oliveira, Hanlin Ren, and Rahul Santhanam. Polynomial-time pseudodeterministic construction of primes. *arXiv preprint arXiv:2305.15140*, 2023.

[16] Lijie Chen, Xin Lyu, Avishay Tal, and Hongxun Wu. New PRGs for unbounded-width/adaptive-order read-once branching programs. In *Proc. 50 International Colloquium on Automata, Languages and Programming (ICALP)*, volume 261 of *LIPIcs*, pages 39:1–39:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[17] James Cook and Ian Mertz. Catalytic approaches to the tree evaluation problem. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 752–760. ACM, 2020.

[18] James Cook and Ian Mertz. Tree evaluation is in space o(log n · log log n). *Electron. Colloquium Comput. Complex.*, TR23-174, 2023.

[19] Stephen A. Cook, Pierre McKenzie, Dustin Wehr, Mark Braverman, and Rahul Santhanam. Pebbles and branching programs for tree evaluation. *ACM Trans. Comput. Theory*, 3(2):4:1–4:43, 2012.

[20] Stephen A. Cook and Charles Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM J. Comput.*, 9(3):636–652, 1980.

[21] Lijie Chen, Ron D. Rothblum, and Roei Tell. Unstructured hardness to average-case randomness. In *Proc. 63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 429–437, 2022.

[22] Lijie Chen and Roei Tell. Hardness vs randomness, revised: Uniform, non-black-box, and instance-wise. In *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 125–136, 2021.

[23] Lijie Chen and Roei Tell. Simple and fast derandomization from very hard functions: Eliminating randomness at almost no cost. In *Proc. 53st Annual ACM Symposium on Theory of Computing (STOC)*, pages 283–291, 2021.

[24] Lijie Chen and Roei Tell. Guest column: New ways of studying the **BPP** = **P** conjecture. *ACM SIGACT News*, 54(2):44–69, 2023.

[25] Lijie Chen, Roei Tell, and Ryan Williams. Derandomization vs refutation: A unified framework for characterizing derandomization. In *Proc. 64 Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2023. To appear.

[26] Holger Dell, Valentine Kabanets, Dieter van Melkebeek, and Osamu Watanabe. Is Valiant-Vazirani's isolation probability improvable? *Computational Complexity*, 22(2):345–383, 2013.

[27] Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. Nearly optimal pseudorandomness from hardness. *Journal of the ACM*, 69(6):1–55, 2022.

[28] Dean Doron, Edward Pyne, and Roei Tell. Opening up the distinguisher: A hardness to randomness approach for **BPL** = **L** that uses properties of **BPL**. In *Proc. 56th Annual ACM Symposium on Theory of Computing (STOC)*, 2024.

[29] Dean Doron, Edward Pyne, Roei Tell, and Ryan Williams. When connectivity is hard, random walks are easy with non-determinism. In *Proc. 57th Annual ACM Symposium on Theory of Computing (STOC)*, 2025.

[30] Dean Doron and Roei Tell. Derandomization with minimal memory footprint. In *Proc. 38 Annual IEEE Conference on Computational Complexity (CCC)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[31] Michael A. Forbes and Zander Kelley. Pseudorandom generators for read-once branching programs, in any order. In *Proc. 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 946–955, 2018.

[32] Bill Fefferman, Ronen Shaltiel, Christopher Umans, and Emanuele Viola. On beating the hybrid argument. *Theory of Computing*, 9:809–843, 2013.

[33] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, New York, NY, USA, 2008.

[34] Oded Goldreich. In a world of **P** = **BPP**. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay Randomness and Computation*, pages 191–232, 2011.

[35] Oded Goldreich. A sample of samplers: A computational perspective on sampling. In Oded Goldreich, editor, *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, volume 6650 of *Lecture Notes in Computer Science*, pages 302–332. Springer, 2011.

[36] Oded Goldreich. Two comments on targeted canonical derandomizers. *Electronic Colloquium on Computational Complexity: ECCC*, 2011.

[37] Uma Girish, Ran Raz, and Wei Zhan. Is untrusted randomness helpful? In *Proc. 14 Conference on Innovations in Theoretical Computer Science (ITCS)*, volume 251 of *LIPIcs*, pages 56:1–56:18, 2023.

[38] Zeyu Guo. Randomness-efficient curve samplers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 575–590. Springer, 2013.

[39] Oded Goldreich, Salil Vadhan, and Avi Wigderson. Simplified derandomization of BPP using a hitting set generator. In *Studies in complexity and cryptography*, volume 6650 of *Lecture Notes in Computer Science*, pages 59–67. Springer, Heidelberg, 2011.

[40] Anna Gál and Avi Wigderson. Boolean complexity classes vs. their arithmetic analogs. *Random Struct. Algorithms*, 9(1-2):99–111, 1996.

[41] Oded Goldreich and Avi Wigderson. On pseudorandomness with respect to deterministic observers. *Electron. Colloquium Comput. Complex.*, TR00-056, 2000.

[42] Oded Goldreich and David Zuckerman. Another proof that $BPP \subseteq PH$ (and more). In Oded Goldreich, editor, *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, volume 6650 of *Lecture Notes in Computer Science*, pages 40–53. Springer, 2011.

[43] Shuichi Hirahara. Non-black-box worst-case to average-case reductions within *NP*. *SIAM Journal on Computing*, 52(6):FOCS18–349–FOCS18–382, 2023.

[44] William M. Hoza. Recent progress on derandomizing space-bounded computation. *Bull. EATCS*, 138, 2022.

[45] Russell Impagliazzo, Ronen Shaltiel, and Avi Wigderson. Extractors and pseudo-random generators with optimal seed length. In *Proc. 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 2000.

[46] Russell Impagliazzo and Avi Wigderson. **P** = **BPP** if **E** requires exponential circuits: derandomizing the XOR lemma. In *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 220–229, 1997.

[47] Russell Impagliazzo and Avi Wigderson. Randomness vs. time: De-randomization under a uniform assumption. In *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 734–743, 1998.

[48] Michal Koucký, Ian Mertz, Edward Pyne, and Sasha Sami. Collapsing catalytic classes. *CoRR*, abs/2504.08444, 2025.

[49] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

[50] Clemens Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4):215–217, 1983.

[51] Yanyi Liu and Rafael Pass. Characterizing derandomization through hardness of levin-kolmogorov complexity. In *Proc. 37 Annual IEEE Conference on Computational Complexity (CCC)*, 2022.

[52] Jiatu Li, Edward Pyne, and Roei Tell. Distinguishing, predicting, and certifying: On the long reach of partial notions of pseudorandomness, 2024.

[53] Pinyan Lu, Jialin Zhang, Chung Keung Poon, and Jin-yi Cai. Simulating undirected *st*-connectivity algorithms on uniform JAGs and NNJAGs. In *Proceedings of 16th International Symposium on Algorithms and Computation (ISAAC)*, volume 3827 of *Lecture Notes in Computer Science*, pages 767–776. Springer, 2005.

[54] Ian Mertz. Reusing space: Techniques and open problems. *Bulletin of EATCS*, 141(3), 2023.

[55] Peter Bro Miltersen and N. V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. *Computational Complexity*, 14(3):256–279, 2005.

[56] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[57] Noam Nisan. **RL** $\subseteq$ **SC**. *Computational Complexity*, 4:1–11, 1994.

[58] Noam Nisan and Avi Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[59] Chung Keung Poon. Space bounds for graph connectivity problems on node-named jags and node-ordered jags. In *34th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 218–227. IEEE Computer Society, 1993.

[60] Edward Pyne, Ran Raz, and Wei Zhan. Certified hardness vs. randomness for log-space. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, 2023.

[61] Edward Pyne and Roei Tell. Composing low-space algorithms. *Electronic Colloquium on Computational Complexity: ECCC*, 2025.

[62] Edward Pyne. Derandomizing logspace with a small shared hard drive. In *Proc. 39th Annual IEEE Conference on Computational Complexity (CCC)*, pages 4:1–4:20, 2024.

[63] Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000.

[64] Alexander Russell and Ravi Sundaram. Symmetric alternation captures BPP. *Comput. Complex.*, 7(2):152–162, 1998.

[65] Michael E. Saks. Randomization and derandomization in space_bounded computation. In Steven Homer and Jin-Yi Cai, editors, *Proceedings of the Eleveth Annual IEEE Conference on Computational Complexity, Philadelphia, Pennsylvania, USA, May 24-27, 1996*, pages 128–149. IEEE Computer Society, 1996.

[66] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[67] Michael Sipser. A complexity theoretic approach to randomness. In *Proc. 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 330–335, 1983.

[68] Michael Sipser. Expanders, randomness, or time versus space. *Journal of Computer and System Sciences*, 36(3):379–383, 1988.

[69] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

[70] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *Journal of the ACM*, 52(2):172–216, 2005.

[71] Ronen Shaltiel and Christopher Umans. Low-end uniform hardness vs. randomness tradeoffs for AM. In *Proc. 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 430–439, 2007.

[72] Ronen Shaltiel and Emanuele Viola. On hardness assumptions needed for "extreme high-end" PRGs and fast derandomization. In *Proc. 13 Conference on Innovations in Theoretical Computer Science (ITCS)*, 2022.

[73] Rahul Santhanam and R. Ryan Williams. On medium-uniformity and circuit lower bounds. In *Proc. 28th Annual IEEE Conference on Computational Complexity (CCC)*, pages 15–23. IEEE, 2013.

[74] Michael E. Saks and Shiyu Zhou. $\mathbf{BP_HSPACE}[S] \subseteq \mathbf{DSPACE}[S^{3/2}]$. *Journal of Computer and System Sciences*, 58(2):376–403, 1999.

[75] Luca Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4):860–879, 2001.

[76] Amnon Ta-Shma, Christopher Umans, and David Zuckerman. Lossless condensers, unbalanced expanders, and extractors. *Combinatorica*, 27(2):213–240, 2007.

[77] Luca Trevisan and Salil Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):331–364, 2007.

[78] Amnon Ta-Shma, David Zuckerman, and Shmuel Safra. Extractors from Reed-Muller codes. *Journal of Computer and System Sciences*, 72(5):786–812, 2006.

[79] Yoav Tzur. Notions of weak pseudorandomness and GF(2^n)-polynomials. Master's thesis, Weizmann Institute of Science, 2009.

[80] Christopher Umans. Pseudo-random generators for all hardnesses. *Journal of Computer and System Sciences*, 67(2):419–440, 2003.

[81] Dieter van Melkebeek and Gautam Prakriya. Derandomizing isolation in space-bounded settings. *SIAM J. Comput.*, 48(3):979–1021, 2019.

[82] Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986.

[83] Avi Wigderson. $nl/\text{poly} \subseteq \oplus l/\text{poly}$. In *Proc. 9th Conf. Structure in Complexity Theory*, 1994.

[84] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. In *Proc. 42nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 231–240, 2010.

[85] R. Ryan Williams. Simulating time with square-root space. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 13–23. ACM, 2025.

[86] Ryan Williams. Personal communication, 2025.

[87] Andrew C. Yao. Theory and application of trapdoor functions. In *Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.

A Alternative proofs for classical theorems using D2P

The notion of D2P gives some nice proofs of previously known results. We cover two proofs from [52]: how can we estimate the expectation of a circuit in the polynomial hierarchy, and how can we solve derandomization of algorithms with two-sided error using only hitting sets?

Derandomizing with games. The strongest known simulation of **BPP** in the polynomial-time hierarchy, first due to Russell and Sundaram [64] (following [50, 67]), asserts that $\mathbf{BPP} \subseteq \mathbf{S_2P}$. The delightful class $\mathbf{S_2P}$ is a subclass of $\Sigma_2\mathbf{P} \cap \Pi_2\mathbf{P}$, where the $\exists$ and $\forall$ players can ignore each other:

Definition A.1. A language $L \in \mathbf{S_2P}$ if there is a polynomial-time algorithm $V(x, g_0, g_1)$ (called the verifier) and a polynomial $p(n)$ such that:

- For $x \in L$, there exists $g_1 \in \{0,1\}^{p(n)}$ (i.e. the strategy of the YES-player) such that for every $g_0 \in \{0,1\}^{p(n)}$ (i.e. the strategy of the NO-player), $V(x, g_0, g_1) = 1$.
- For $x \notin L$, there exists $g_0 \in \{0,1\}^{p(n)}$ (i.e. the strategy of the NO-player) such that for every $g_1 \in \{0,1\}^{p(n)}$ (i.e. the strategy of the YES-player), $V(x, g_0, g_1) = 0$.

Theorem A.2. *There is a* $\mathbf{S_2P}$ *protocol to estimate* $\mathbb{E}[D] \pm (1/3)$ *for a circuit* D.

Proof Sketch. We want strategies for both players that ensure the other cannot convince the verifier that the expectation differs from the true value.

Lets assign Player 1 the job of giving a distribution $\mathcal{S}$ that approximates the expectation for *every* small circuit (which in particular includes the circuit D given to both players). But of course, Player 1 can lie and give a distribution that does not approximate $\mathbb{E}[D]$. How can Player 2 make sure that no distribution that misleads on the value of $\mathbb{E}[D]$ sneaks through? By giving a

valid D2P transformation for D! If P1 gives a distribution $\mathcal{S}$ with no small predictors (which exists via a simple counting argument), no matter what predictors P2 writes down, $\mathcal{S}$ will be accepted and used to estimate $\mathbb{E}[D]$, and since a distribution with no small predictors has no small distinguishers either, that estimate will be accurate. On the other hand, if P2 gives a valid D2P transform for D, by the definition of D2P transformation any distribution $\mathcal{S}'$ that is not predicted must do a good job estimating $\mathbb{E}[D]$. Since a distribution $\mathcal{S}$ with no small predictors and a valid D2P transform for D always exist, both players can ignore each other. □

Hitting sets imply derandomization of algorithms with two-sided error. A similar construction gives a new proof that one-sided derandomization implies two-sided derandomization. Recall that in one-sided derandomization we are given a circuit $D : \{0,1\}^n \to \{0,1\}$ and asked to distinguish between $\mathbb{E}[D] = 0$ and $\mathbb{E}[D] \geq 1/2$. The analogue of a pseudorandom generator (resp., pseudorandom set) in this regime is known as a hitting set generator (reps., hitting set):

Definition A.3. For a class of circuits $\mathcal{D}$ on n bits, a hitting set $H \subseteq \{0,1\}^n$ for $\mathcal{D}$ has the property that for all $D \in \mathcal{D}$ with $\mathbb{E}[D] \geq 1/2$, there is $y \in H$ where $D(y) = 1$.

While hitting sets are not obviously capable of giving two-sided derandomization, classical results show that they do [2, 3, 8, 39, 42, 50, 67]. In [52] they showed that using D2P, we can get a new, relatively simple proof for this fact.

To do so, they use an algorithm of Goldreich and Widgerson [41] (slightly generalized in [52]) that gets a collection of next-bit predictors on $\{0,1\}^n$ (which we can think of as a D2P transform for D), runs in *deterministic* polynomial time, and outputs a distribution $\mathcal{S}$ that is unpredictable to all of them:

Theorem A.4 (Informal, see [41] and Lemma 4.2 [52])**.** *There is a deterministic polytime algorithm that given a collection of next-bit-predictors $P_1, \ldots, P_{\mathrm{poly}(n)}$ over $\{0,1\}^n$, outputs a distribution $\mathcal{S}$ of size at most n^5 such that no P_i is a $(1/3n)$-predictor for $\mathcal{S}$.*

We can then use Theorem A.4 to prove the result.

Theorem A.5. *Suppose there is an hitting set for circuits of size n on n input bits that can be constructed in time* poly(n)*. Then there is a polynomial-time algorithm that, given a circuit $D : \{0,1\}^n \to \{0,1\}$, estimates $\mathbb{E}[D]$ to additive error $1/3$.*

Proof Sketch. We will show that using H, we can produce a D2P transform $\mathcal{P} = (P_1, \ldots, P_t)$ for D that is secure against all distributions of size at most n^5, in the sense that every small distribution that D distinguishes from random is predicted by some $P \in \mathcal{P}$. We then produce a distribution $\mathcal{S}$ that is unpredictable to all these predictors via Theorem A.4, and use that distribution to estimate $\mathbb{E}[D]$. Since the D2P transformation is secure, this estimate must be accurate. Details follow.

Let $\mathbf{w}$ be an arbitrary distribution of size n^5, and suppose D is a $(1/3)$-distinguisher for $\mathbf{w}$. By Proposition 4.8, a random string z, b, j of length $\widetilde{O}(n)$ has the property that the function

$$P_{(z,b,j)}(x_{<j}) = D(x_{<j} \circ z_{\geq j}) \oplus b$$

is a $(1/3n)$-predictor for $\mathbf{w}$, with probability $\rho = 1/\mathrm{poly}(n)$. Moreover, there is a circuit $\mathrm{TEST}_{D,\mathbf{w}}(y, b, j)$ of size $\widetilde{O}(n^6)$ that checks if the string (z, b, j) creates such a good predictor. We may assume by a standard error reduction that the hitting set H hits all size-$\widetilde{O}(n^6)$ circuits with expectation at least ρ, so if we consider the collection of predictors obtained by enumerating all elements of the hitting set

$$\mathcal{P} = \{P_{y=(z,b,j)} : y \in H\}$$

there must be some $P \in \mathcal{P}$ where P is a $(1/3n)$-predictor for $\mathbf{w}$. Thus, for *every* bad distribution of size at most n^5, there is $P \in \mathcal{P}$ that is a $(1/3n)$-predictor, and there are $|H| = \text{poly}(n)$ total such predictors.

We invoke the algorithm of Theorem A.4 on this collection $\mathcal{P}$. The distribution $\mathcal{S}$ we receive as output is of size n^5 and $(1/3n)$-unpredictable to all $P \in \mathcal{P}$, so it must be the case that

$$\left| \mathop{\mathbb{E}}_{y \in \mathcal{S}}[D(y)] - \mathbb{E}[D] \right| \leq 1/3$$

and we can return $\mathbb{E}_{y \in \mathcal{S}}[D(y)]$ as our estimate. □

A related proof idea implies that *white-box* one-sided derandomization (i.e. an algorithm that given D returns if $\mathbb{E}[D] = 0$ or $\mathbb{E}[D] \geq 1/2$, with no correctness guarantees otherwise) likewise implies two-sided derandomization. Unfortunately, like all other proofs of this fact, if the non-black-box derandomization runs in time $T(n) \gg \text{poly}(n)$, the corresponding two-sided derandomization runs in time $T(T(n)) \gg T$. For full results, see [52, Theorem A.2].

The Machine Learning Column

BY

Pablo Barceló

Pontificia Universidad Católica de Chile
Avda. Vicuña Mackenna 4860
Santiago, Chile
pbarcelo@uc.cl

AND

David Saulpic

Institut de Recherche en Informatique Fondamentale (IRIF)
CNRS & Université Paris Cité
8 Place Aurélie Nemour
75013 Paris, France
david.saulpic@irif.fr

BEATCS no 148

Explaining Hallucinations from a TCS perspective: Interview with Santosh Vempala

Abstract

After sketching the landscape of how TCS can or does contribute to ML in the last Bulletin, we propose here to dive into one of the most recent attempts to shed a TCS light on some ML phenomenon – namely, using statistics to explain hallucination in language generation. We conducted for this an interview with Santosh Vempala, who pioneered this direction with Adam Tauman Kalai. Santosh tells us here the story behind the maths: the motivation for the paper, how they proceed with modeling and what are the implications of the result.

1 Introduction

For this edition of the column, we had the chance to interview Santosh Vempala about two recent papers on hallucinations in large language models (LLMs): the STOC 2024 paper *Calibrated Language Models Must Hallucinate* (with Adam Tauman Kalai) [4], and its follow-up *Why Language Models Hallucinate* (with Kalai, Ofir Nachum, and Edwin Zhang) [3].

These papers caught our attention because they are a rare attempt to bring theoretical computer science (TCS) tools to bear on a socio-technical phenomenon that has quickly become central in practice. Doing so requires more than a standard "theorem–proof" contribution: one must decide what to formalize, what to abstract away, and which simplifying assumptions remain faithful to the real object of study. We therefore wanted to discuss the modeling process itself—the choices made, the alternatives left on the table, and how these choices shape both the interpretation of the results and the kinds of follow-up questions that become tractable. We briefly introduce Santosh and summarize the main messages of the two papers before turning to our conversation.

Santosh Vempala is the Frederick P. Storey II Chair of Computing and a Distinguished Professor in the School of Computer Science at the Georgia Institute of Technology. His research sits at the intersection of algorithms, randomness, high-dimensional geometry, and the foundations of data science, with influential

work on sampling and optimization for high-dimensional distributions. He received the 2024 Delbert Ray Fulkerson Prize (jointly with Ben Cousins) for work on algorithms for computing volume and Gaussian volume, and he is a Fellow of the ACM and of the American Mathematical Society. Beyond research, he has played an active community and institution-building role at Georgia Tech, including leadership in interdisciplinary theory programs (notably the Algorithms, Combinatorics, and Optimization Ph.D. program) and the creation of the *Computing for Good* (C4G) initiative, which connects computing ideas to societal challenges through project-based work.

2 Calibrated Language Models Hallucinate

The first of the two papers, "Calibrated language models must hallucinate", traces the origin of hallucinations to a purely statistical property called *calibration*. Doing so requires introducing a number of definitions.

- First, what is a language model? From a TCS perspective, this is simply a probability distribution over strings. Indeed, a "next-token predictor" assigns, for each position, a probability distribution to the next token conditioned on the previous ones. This is mathematically equivalent to specifying a single probability distribution over all finite strings of tokens.

- Second, what is a hallucination? The authors adopt a deliberately simple setting. There is a set of "facts" F contained in a universe of possible strings Y. The hallucination rate is the total probability mass that the model assigns to strings that are not facts, that is, to elements of $Y \setminus F$.

- In addition, "the world is assumed to contain randomness": there is an underlying distribution p over the set of facts. The training data $O \subseteq F$ is generated by sampling from this distribution.

Hence, the model is trained by observing only a subset of all existing facts. A concrete example is the set of birthdays of TCS researchers. Some birthdays are frequently mentioned, others rarely, and many may never appear in public data at all. The distribution p is highly non-uniform, and any model can only be trained on a partial view of these facts.

Now, calibration is the following property. A distribution $\hat{p}$ produced by the model is calibrated with respect to the true distribution p if, for every probability value $y \in [0, 1]$, all elements that the model predicts with probability y indeed appear, on average, with probability y under p; formally,

$$\mathbb{E}[\, p(x) \mid \hat{p}(x) = y \,] = y.$$

Intuitively, if a weather model predicts rain with probability 30% on a collection of days, then it should indeed rain on about 30% of those days. In the interview below, Santosh Vempala provides a different intuition on calibration and why it is desirable.

The second paper relaxes this notion. Instead of requiring calibration everywhere, it only requires calibration on the set of *frequent* elements. Formally, letting

$$\mathcal{A} := \{x : \hat{p}(x) \geq 1/|Y|\}$$

be the set of elements predicted with above-uniform probability, the requirement is simply that

$$|p(\mathcal{A}) - \hat{p}(\mathcal{A})| \tag{1}$$

is small.

The main technical result of the first paper is a lower bound on the hallucination rate of a language model whose calibration error is small. As Santosh explains below, this assumption is natural: the standard pretraining objective, which minimizes *log-loss*[1], forces the model to be approximately calibrated. Indeed, the paper shows that any model with small log loss necessarily has small calibration error. Thus, assuming low miscalibration accurately reflects the behavior of pretrained base models.

Two further assumptions complete the picture. First, conditioned on the observed training data O, all unobserved facts in $F \setminus O$ are assumed to be equally likely to be true. Formally, for all $y, y' \in F \setminus O$,

$$\Pr[y \in F \mid O] = \Pr[y' \in F \mid O].^2$$

Second, the universe of possible statements is assumed to be much larger than the set of true facts: there exists a (large) s such that $|F| \leq e^s |Y \setminus F|$. In the birthday example, for each researcher there is a single correct date and 364 incorrect ones, yielding $s = \ln(364)$.

Under these assumptions, the main conclusion can be stated informally as follows: if not all facts are observed during training and the model is calibrated, then the hallucination rate must be at least the total probability mass of the unobserved facts. The intuition is simple. Calibration forces the model to assign approximately the correct total probability mass to the observed facts O. The remaining probability mass must therefore be assigned to unobserved statements. But among these, the model has no statistical way to distinguish true facts from false ones.

[1]The log-loss is $\mathbb{E}_{x \sim p_{train}} - \log \hat{p}(x)$, where p_{train} is the training distribution and $\hat{p}$ is the one produced by the model. It has been observed, e.g. in [1], that neural networks trained by minimizing the log-loss are calibrated – although for a different notion of calibration than the one used here.

[2]This equality can be relaxed to hold up to a multiplicative factor r, which then appears in the final bound. For simplicity we present only the symmetric case.

Since incorrect statements vastly outnumber correct ones and are equally likely conditioned on O, most of this remaining mass will inevitably fall on hallucinations. Thus the hallucination rate is close to $1 - p(O)$.

The first paper makes this quantitative by relating the unobserved mass to the number of facts that appear exactly once in the training data, via the so-called Good–Turing (missing-mass) estimator. The second paper takes a different route. Instead of estimating the missing mass, it reduces a standard binary classification problem to language generation. In this reduction, the model is asked to generate statements and is then evaluated by a simple classifier that decides whether a generated statement is valid or invalid. The key observation is that any hypothesis with large error on this *Is-It-Valid* (IIV) classification problem must necessarily assign significant probability mass to invalid statements when used as a generator. In this way, lower bounds for supervised classification error translate directly into lower bounds on hallucination rates, independently of any assumptions about missing mass.

3 The Interview

We present here a transcript of an interview we had with Santosh Vempala over zoom. We slightly edited it only to cite the papers mentioned and explain a few technical terms in footnote. The summary above also presents the general context, and explains the contribution we are discussing below with Santosh.

DS: Our initial questions are sort of an introduction to your papers and the way you thought about it. How did you start working on this project? And in your opinion, what's the contribution of the two papers?

Santosh Vempala: I have known my brilliant friend and co-author Adam Kalai for a long time. He was a few years junior to me in graduate school at Carnegie Mellon, and also he was a postdoc with me at MIT for two years. We've been discussing various aspects of theory over the years, but a couple of years ago we started talking more about language models—and in particular, why they behave the way they do.

We had the same advisor, Avrim Blum, who's a learning theorist. While that is not my primary focus, I certainly think about problems coming from high-dimensional learning and things like this. Adam is a leading researcher in learning theory. In fact, he was so worried about AI safety—this was already two years ago—that he moved from Microsoft to OpenAI, despite having offers from Princeton and CMU.

For me, the main question at the time was to explain some of the crazy things LLMs were doing consistently, which I still don't fully understand today, like

producing correct grammar. Even though you're only trained on next-token prediction, why are you getting this long-range structure—or "learning," in quotes—and so on?

At the same time, there were mistakes people would make fun of—like arithmetic mistakes—but there were also other mistakes happening, and bad advice on various topics. That seemed like an easier question to define. You can just say there's a labeling and you're outputting things with the wrong label: valid/invalid, true/false, whatever you want to call it depending on context.

So that's how we started. It took a while. We definitely began with just "facts," because that special case was fully well-defined: every document is a single statement and either it's a fact or a hallucination. It's two labels. You are only trained on facts—will you still produce hallucinations? After some iterations, we realized that as long as you're calibrated, the answer is yes. That's how it started. It also took a while to write the paper, only after we found this connection to the Turing estimate, we thought, okay, this is something nice and worth writing about.

DS: And for which community is this paper meant? Is it for TCS, for a general audience, or for statistics ?

Santosh Vempala: Yeah, so the first paper: the lower bound is just one single statistical lower bound, basically. It has nothing to do with the model or the architecture. It's really just a statistical statement, combining two things: calibration (which is classical, from statistics and game theory) and an argument about error rates.

We were motivated by practical reasons, and the theory came out not too messy. And of course there's always been this question in the background: what can theory do for ML? How can we analyze these things? It's persisting. We figured, okay, let's see if STOC will think it's sufficiently interesting and so on. In this case, they were fine.

PB: Do you see this as some kind of no-free-lunch theorem?

Santosh Vempala: The surprising thing is that *a priori* you might not have thought that being calibrated and being truthful (or not hallucinating) are at odds. Usually with trade-offs like no-free-lunch, there are two things that intuitively are in tension and then you prove a conservation-type statement: you gain one, you lose the other. But here, if you are getting more accurate, you'd think you should hallucinate less. That's the naive high-level surprise. Once you look into the actual definitions, you see it's not that simple.

DS: You said that being calibrated means being more accurate. Can you explain this—and explain why calibration is good for predictions?

Santosh Vempala: Calibration is very general. Let me give you a view that you may not see in the literature. Take any probability distribution over some set of

objects (finite, for now). Each element has some probability, summing to one. Call that the "ground truth" distribution.
Now, for another distribution to be calibrated with respect to this one, it can be any *coarsening*. By that I mean: take the original distribution, partition the elements into groups (some singleton groups, some large groups—whatever), and within each group, assign to every element the average of the true probabilities in that group. That coarsened distribution is certainly calibrated. And the other direction is also true: any calibrated distribution must be a coarsening.
The most trivial calibrated distribution is: take all elements and replace everyone's probability by the overall average. You're "accurate" in a weak sense: you get the overall average right. You can strengthen this by requiring the average to be correct on different parts (e.g., things you predict with probability at least $1/2$ versus at most $1/2$). The strongest version is the "bucket" version: for any value the model outputs, say 0.2, look at all elements for which it outputs 0.2; the true average probability over that bucket should also be 0.2. Like weather: if you predict 10% rain on all days in a bucket, it should rain on 10% of those days. You can apply this to anything where there's a distribution—including documents as outcomes.

DS: I had a question about the process of taking averages: if you choose groups in the wrong way, the average could be very far from the original distribution.

Santosh Vempala: It depends how you measure "far away." It turns out that if you look at *log loss*, coarsening cannot worsen it: the log loss with respect to the training distribution cannot get worse.
In the second paper we show something simpler: we don't need calibration in arbitrary bins. You just need two bins: low-probability and high-probability.[3] In each bin you should be calibrated in the sense that your average matches the ground-truth average.
Now, if you minimize log loss and reach any local minimum — not necessarily the global minimum! — then this calibration error is zero. Log loss is essentially what base models optimize before post-training. In fact, what we prove is that the gradient of the log loss is exactly the calibration term that appears in the theorem.

DS: Interesting, thanks! Our next question was precisely about this theorem, in the second paper: you have a calibration term related to the high-frequency predictive elements, and you also have this "is-it-valid" error term. How typical are models with high "is-it-valid" error?

Santosh Vempala: I see what you mean—the ability to distinguish valid from invalid. The nice thing about the reduction is that it's general; it's not specific to any particular failure mode. The error could be bad for statistical reasons (those are

[3]This calibration is the term defined in Equation (1)

the ones we can lower bound, like the "arbitrary facts" setting). But it could also be bad because of a bad model—like early GPT versions doing arithmetic poorly. That's not statistical hardness; there are short rules for addition and multiplication, but the model fails anyway.
Or it could be computational complexity: e.g., completing a prompt that effectively asks for factoring. That's hard not because of statistics, but because it's computationally hard.
So it's a bit hard to answer in the abstract. In practice, based on base-model hallucination rates today (on internet data or domain-specific data), it seems quite high.

PB: Suppose I fix the training distribution. Your result says: for any hypothesis, the error is at least this "is-it-valid" error minus some terms. But if my hypothesis *is* the training distribution, then the error is zero. So can you quantify: for how many hypotheses is the error large? If I pick a hypothesis randomly, would the error typically be large? This feels related to the first paper, where you say there are many hypotheses for which the error is large.

Santosh Vempala: Right—but that depends on the setting. For things that are easy to classify (say, data separable by a halfspace), many reasonable learners will achieve small classification error. Here the statement is: you give me a model, an architecture, a training procedure, and you output some hypothesis. Compare it to the true distribution. The probability of generating nonsense (hallucinations) is lower bounded by your ability to solve the corresponding supervised classification problem (valid vs. invalid), minus the calibration term. And the reduction uses only a very simple classifier derived from your model: a probability threshold. If that threshold classifier's error is high, you will hallucinate.
So it's like a worst-case reduction: for each model, there's a direct reduction—a classical reduction.

DS: So should we see this "is-it-valid" error as a simpler proxy term, easier to compute, that still gives a bound?

Santosh Vempala: Yes. And it's a thing we understand in learning theory: supervised classification. We know sources of error—VC dimension/sample complexity, and sometimes computational hardness (e.g., in restricted models). The point is not new insight on supervised classification; it's connecting that to generation/hallucination.

PB: Why couldn't it be that some optimization method leads you to a good hypothesis anyway—one with small error? How do we know we won't just end up with something bad?

Santosh Vempala: Two things. First: if what you come up with is *not calibrated*, then we say nothing.

PB: That's one of our later questions, but it's important here too. So you don't know what happens then?

Santosh Vempala: Right. For example, a model could just say "I don't know" every time. Or it could say something true all the time—"the sun rises in the east"—regardless of the question. Then it's not hallucinating (in a sense), but it's not meaningfully responding, and calibration can be very bad.
But if you have some control on calibration error, then the theorem applies. It doesn't matter how you trained, what optimization method you used, what architecture you chose: if at the end your model has bounded calibration error, then the hallucination rate is at least the validity-classification error minus the calibration term.
And one nice thing is that this condition is natural for modern base models: during pre-training, everyone minimizes log loss (equivalently *KL divergence*[4] to the training distribution). If you go to any local minimum of log loss, you get the kind of calibration we need. So base models will hallucinate roughly as much as their classification error.

DS: We also had a question about the other key assumption: the ratio of true facts versus possible hallucinations. In both papers, that ratio is assumed to be very small—there are far more possible hallucinations than true facts. But in structured domains like mathematics, formal proofs, and programming languages, maybe that's not the case. What do you expect the lower bounds to become there?

Santosh Vempala: If the numbers are equal—say half valid, half invalid—then the original main theorem says nothing because the term involving V/E becomes trivial. But two points.
First, there is a refinement where we can trade off the leading term (the validity error) with the ratio term. You can replace V/E with something like $V/(V+E)$ so you're subtracting the true fraction rather than V/E. That is best possible, since a trivial classifier can always guess "valid" and be correct with probability $V/(V+E)$. But you pay a price: the leading term gets multiplied by a factor depending on V and E. It's a partial trade-off and not fully satisfying, so we did not include it.
Second, if you look at Theorem 3 in the new paper (pure multiple choice)[5], we only need that there is at least one wrong answer. You get a lower bound even for true/false questions: one correct answer, one wrong answer. The original theorem

[4]The KL (for Kullback-Leibler) divergence between two distribution p and q is defined as $\sum_x p(x) \log \frac{p(x)}{q(x)}$. It is not a proper distance, but is commonly used to measure a loss in using Q instead of P. Models in ML are often trained to minimize the KL divergence between the ground-truth distribution and the one learned by the model.

[5]This theorems relates the hallucination rate to the optimal misclassification rate for a simple family of predictors ; in particular, it gives a *multiplicative* bound without any additive corrective terms, as opposed to the other results presented.

says nothing when $V = E$, but Theorem 3 still gives a nontrivial bound (basically a constant fraction). It's also the simplest theorem in the paper and doesn't even need calibration.
So we do not yet have the full answer for the trade-off as V approaches E and E goes to 0.

DS: One more question about modeling, perhaps the main one. The proofs aren't too complicated; so to me the contribution is the modeling—how you chose assumptions and abstracted away training details. That seems like a very nontrivial process. If I want to work on this, where do I even start? What can you say to help?

Santosh Vempala: That took a long time. In some ways it takes longer than solving a hard but well-defined problem. It's easy to go down paths where assumptions are messy—or worse, assumptions are clean and proofs are nice but irrelevant to the original motivation.
Luckily, my co-author—especially between the first and second paper—was at OpenAI. He moved there for personal reasons because he's worried about AI safety, and he always kept it grounded. I was happy to go in directions relaxing assumptions or making proofs more complicated, but he would keep bringing us back: "I don't think this actually means anything." Having that kind of empirical sanity check—from someone embedded in the field—was very helpful (he also has a knack for generating simple proofs!)

PB: Your first paper has been highly cited—I think over 200 citations, which is impressive. Many citations come from machine learning more than theory. How do you feel the ML community, or even OpenAI, received these results?

Santosh Vempala: The first paper was written when Adam was at Microsoft and we posted it before he joined OpenAI, so that was fine.
ML people are definitely interested. It's such an exciting time that they often don't have time to sit back and do theory—they need to keep going because things are moving so fast.
Between the first and second paper we kept thinking how to make it better and understand more. We had the basic results of the second paper several months before posting. Then there was this OpenAI angle, because they were publicly acknowledging hallucinations as a serious problem. Adam helped them write a blog post. Once OpenAI puts something on its front page, thousands of ML people read it or at least become aware of it.
They decided to take ownership: they said it's a real and serious problem, and even reminded us that humility is one of OpenAI's core values. Besides the science, that may have been one of the most useful outcomes: acknowledging the problem and working on it full-time.

For example, by mid-October, if you asked for the birthday of someone who doesn't have it online, GPT stopped making it up and instead asked for more information. Before mid-October, it would happily invent a date—and invent a different one if you asked again. They also reduced making up citations: now it's much harder to get a fake citation unless you try to break the system.

DS: Do you know whether they stopped hallucinating in other examples not in the paper, or just the ones everyone tried because they were in the paper?

Santosh Vempala: Historical facts is one. Another is made-up citations—they've been spending a lot of time on that. The rate is much lower now. If you behave like a normal user, it's not making up citations.

PB: Since we're entering philosophical questions: have you received criticism like, "LLMs aren't really learning in a pure distributional way; they're learning structure, they learn something about facts," etc.? How do you respond?

Santosh Vempala: There's a range of criticisms. Nothing too acerbic. Everything from "this is obvious" to "everything an LLM produces is a hallucination by definition because it's a statistical model"—a philosophical take.
Another common jump is: "they don't understand structure," "they don't reason," therefore "of course they hallucinate." I think that's an oversimplification.
There's a fundamental question: can a purely statistically trained machine reason or do symbolic manipulation without specialized training? I haven't seen a lower bound that says: if you only do statistical training, you *must* fail at reasoning on some task. If that were true, you'd want a statement like that. I haven't seen one.

PB: Probably it's quite difficult to formalize.

Santosh Vempala: Yeah—and maybe it's not true.
Let me add one thing about understanding and reasoning. Understanding is hard to define, but reasoning—logical deduction—we can at least talk about. For example, mathematical proof. Suppose you train on sufficiently many correct proofs: will the model then produce correct mathematical statements and proofs?
Just last month we put out a preprint called *The Long-Range Benefits of Next-Token Prediction* [2]. We prove a bound on model size ensuring that, with respect to any distinguisher (an algorithm that tries to tell apart outputs of the model from samples of the training distribution), the model can self-improve without knowing the distinguisher, just by minimizing log loss.
The theorem says: by minimizing log loss with a sufficiently large model (polynomial size, with an explicit bound), you reach a model whose outputs cannot be distinguished from the training distribution by distinguishers up to a certain size. In the proofs analogy: proofs generated by the model versus correct proofs are indistinguishable for proofs up to some length k; model size is polynomial in k and in the distinguisher size.

DS: If a human is the distinguisher, what is the "size"?

Santosh Vempala: Good analogy. You need to bound the size of the machine you're using. Think of it as some constant number of nodes. And you have to bound how much you can look ahead—your window size. If the window size is k and your machine size is D, then you need a language model of size about k^2D (and also scaling with $1/\varepsilon^2$, where ε is the indistinguishability parameter).

DS: So k is the size of your own memory?

Santosh Vempala: Yes—the window size: how many tokens you can use.

DS: You compared a statistical goal and a complexity goal. Your paper shows statistical limits on LLMs. Should we move away from statistics—should big companies move away from the statistical view—to reduce hallucinations and still produce novelty? Or is it incomparable?

Santosh Vempala: One thing we mention in the second paper (though we can't say much theoretically) is that post-training reduces hallucinations explicitly, and of course it makes the model less calibrated. But there is no reason in principle why you couldn't drive hallucinations close to zero. People are already doing ad hoc things; we propose some specific directions.
One reason hallucinations persist despite post-training is that evaluations often do not reward "I don't know." It's better (under many reward schemes) to guess: you might get it right and improve your score, whereas "I don't know" is treated as wrong for sure. We also discuss a notion we call "behavioral calibration."
Statistics is a good starting point, but we have to get into computation. That's why the reduction is interesting: it transfers lower bounds whether the source is statistical or computational.

DS: Perhaps your proof suggests calibration is not such a great property to desire. Also: are we calibrated as humans? When we speak and use language, are we calibrated? Is calibration the right thing to target if we want models to behave like humans with respect to facts?

Santosh Vempala: Good. It's kind of amazing that statistical training can produce output that matches the input distribution. It's not just calibrated; it tends to be accurate (low log loss).
For humans: think about young children or babies. They're more calibrated—closer to the input—they reproduce what they've heard. Tone, word choice, even bad language, matches the training distribution. As we get older, there's post-training. What we hear is not necessarily what we say: we filter, post-process, and adapt to context. So humans become less calibrated relative to their initial distribution.

There is also an empirical follow-up paper by Muqing Miao and Michael Kearns [5]. They evaluate the "arbitrary facts" setting, looking at the Turing estimate (fraction of singletons), calibration error, and the other terms, and they find the behavior is close to what the theorem suggests.
So yes: reducing hallucinations tends to reduce calibration, empirically.

DS: So is calibration a good statistical goal?

Santosh Vempala: It's a good starting point, but in many contexts it's not the right endpoint. It helps you see that if you're well calibrated in a domain where you cannot possibly store all facts, then you should worry: you may produce nonsense some of the time.

PB: For the first edition of our column: David and I wrote a short survey of connections between theoretical computer science and machine learning. For you, what should the role of theoretical computer science be for machine learning, and how should these two interact?

Santosh Vempala: That's an important question. I've helped organize several Simons programs: data science, foundations of data science, foundations of machine learning, computation in the brain. Those settings are nice because you get lots of theoreticians, but you also explicitly invite domain experts. Those interactions were quite fruitful.
A lot of what TCS can do is to formulate phenomena from practice—LLMs doing really well, questions of safe AI—more precisely. It's not going to be one question; it's nuanced. But we have a style: go to the simplest version that's still interesting, solve it, get insight, build tools slowly, and move forward. We need to do that even while the field is racing.
Safety is becoming more of a concern. Not necessarily malicious actors, but also failure modes from the machines themselves. The motivation is there for theoreticians to take a step back (which we can afford), formulate problems, develop techniques, and build tools that can be used later. We have a track record: online algorithms, optimization, high-dimensional sampling—practice eventually follows theory. So why not get started on these relevant questions now?

PB: We have time to step back—but the pace of ML is outrageous.

Santosh Vempala: Absolutely. And it's even more overwhelming for non-theoreticians, especially those not in premier labs. They mostly watch and try to use what's coming out.

PB: Yeah. Okay. It's been really helpful, interesting, and fun.

DS: Thank you very much. That was a very nice conclusion—very optimistic for TCS.

Santosh Vempala: Thank you. Thank you for taking the time.

References

[1] Jaroslaw Blasiok, Parikshit Gopalan, Lunjia Hu, Adam Tauman Kalai, and Preetum Nakkiran. Loss minimization yields multicalibration for large neural networks. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, Berkeley, CA, USA, January 30 - February 2, 2024*, volume 287 of *LIPIcs*, pages 17:1–17:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[2] Xinyuan Cao and Santosh S. Vempala. Provable long-range benefits of next-token prediction. *CoRR*, abs/2512.07818, 2025.

[3] Adam Tauman Kalai, Ofir Nachum, Santosh S. Vempala, and Edwin Zhang. Why language models hallucinate. *CoRR*, abs/2509.04664, 2025.

[4] Adam Tauman Kalai and Santosh S. Vempala. Calibrated language models must hallucinate. In *STOC*, pages 160–171, 2024.

[5] Muqing Miao and Michael Kearns. Hallucination, monofacts, and miscalibration: An empirical investigation. *CoRR*, abs/2502.08666, 2025.

THE LOGIC IN COMPUTER SCIENCE COLUMN

BY

ANUJ DAWAR

Department of Computer Science and Technology
University of Cambridge, Cambridge, CB3 0FD, UK
anuj.dawar@cl.cam.ac.uk

In this issue, I am very pleased to introduce a guest column by Gyeongwon Jeong, Seonghun Park and Hongseok Yang of KAIST. I invited Hongseok to contribute this column after hearing a wonderful talk from him last summer, which is based on joint work with his students Gyeongwon and Seonghun. I found the topic to be a fascinating example of logic emerging from mathematical practice. The work is about *flag algebras* which is a framework introduced by Razborov to reason about certain limiting structures in extremal combinatorics. The main concerns that motivate it are not at all logical and have to do with classical questions in combinatorics. Yet, the framework that Razborov created can be understood as a system of formal logic. It is this idea that the authors of the present column have sought to formalize. They define flag algebra with a rigorously presented syntax and semantics in a way that can be implemented in an automated theorem proving system. It makes for very interesting reading.

BEATCS no 148

An Introduction to Razborov's Flag Algebra as a Proof System for Extremal Graph Theory

Gyeongwon Jeong Seonghun Park
Hongseok Yang
School of Computing, KAIST, South Korea.

Abstract

Razborov's flag algebra forms a powerful framework for deriving asymptotic inequalities between induced subgraph densities, underpinning many advances in extremal graph theory. This survey introduces flag algebra to computer scientists working in logic, programming languages, automated verification, and formal methods. We take a logical perspective on flag algebra and present it in terms of syntax, semantics, and proof strategies, in a style closer to formal logic. One popular proof strategy derives valid inequalities by first proving inequalities in a labelled variant of flag algebra and then transferring them to the original unlabelled setting using the so-called downward operator. We explain this strategy in detail and highlight that its transfer mechanism relies on the notion of what we call an adjoint pair, reminiscent of Galois connections and categorical adjunctions, which appear frequently in work on automated verification and programming languages. Along the way, we work through representative examples, including Mantel's theorem and Goodman's bound on Ramsey multiplicity, to illustrate how mathematical arguments can be carried out symbolically in the flag algebra framework.

1 Introduction

Razborov's flag algebra [17] is a framework for reasoning about *asymptotic* (i.e., large-size) properties of combinatorial structures. In extremal graph theory, many problems ask for the maximum or minimum possible density of a fixed finite pattern (such as an edge, a triangle, or an induced cycle) among all large graphs satisfying certain constraints (such as forbidding a subgraph). Flag algebra turns such questions into optimisation problems over a compact space of limit objects,

and provides a symbolic calculus for deriving valid inequalities between densities of those finite patterns.

Since its introduction, the flag algebra framework has led to solutions of, and substantial progress on, several central problems in extremal graph theory, including the minimum triangle density problem [18], Erdős's pentagon problem [11, 8], the induced 5-cycle density problem [2], Turán's tetrahedron problem [19, 1], and the rainbow-triangle density problem [3]. Many key proof steps in the framework have also been automated: FLAGMATIC [6, 16] is a widely used tool that underpins several of these results in extremal graph theory.

This article surveys flag algebra for computer scientists working in logic, programming languages, automated verification, and formal methods. Although several excellent surveys on flag algebra exist [20, 5, 9], they are written primarily for mathematicians, presenting flag algebra as a symbolic calculus for familiar arguments (e.g., double counting and applications of Cauchy–Schwarz). They also emphasise the algebraic viewpoint of flag algebra. By contrast, we take a *logical* perspective and present flag algebra in terms of syntax, semantics, and proof strategies, in a style closer to formal logic. We also highlight that some core constructions (notably the so-called downward operator that relates labelled and unlabelled settings) can be understood via ideas reminiscent of Galois connections and, more generally, categorical adjunctions—concepts that frequently appear in automated verification and programming languages. The results explained in this survey article, such as lemmas and their proofs, are from Razborov's original paper on flag algebra [17], although they are presented here in a slightly different style to emphasise the logical perspective and fit the overall narrative.

The rest of the paper is organised as follows. In the remainder of this introduction, we fix some notation and terminology used throughout the paper. In Section 2, we review the notion of induced subgraph density and use it to formulate representative asymptotic problems in extremal graph theory. In Section 3, we introduce limiting graphs, which enable a concise description of these problems and form the semantic basis of flag algebra. In Section 4, we present flag algebra as a logical system with two kinds of terms—density expressions and assertions—together with a semantics in terms of limiting graphs. We also illustrate, through two examples, how standard extremal arguments can be expressed as derivations in this system. The key steps in both examples rely on a nontrivial inequality. In Section 5, we describe a strategy for proving such inequalities using labelled variants of flag algebra together with a mechanism for transferring inequalities from the labelled setting to the unlabelled one via the so-called downward operator. Finally, in Section 6, we discuss research opportunities in flag algebra for computer scientists working in logic, programming languages, automated verification, and formal methods.

Before we begin, we fix some notation and terminology. For each $n \in \mathbb{N}$, we

write $[n]$ for the set $\{1, 2, \ldots, n\}$. For a set X and a nonnegative integer n, we write $\binom{X}{n}$ for the set of all n-element subsets of X. Throughout the paper, we consider finite, simple, undirected graphs G whose vertex set is a subset of $\mathbb{N}$. Here, *simple* means that G has neither loops (i.e., edges from a vertex to itself) nor multiple edges (i.e., more than one edge between the same pair of vertices). Thus, such a graph G is formally a pair consisting of a finite set $V \subseteq \mathbb{N}$ and a set $E \subseteq \binom{V}{2}$. We refer to these simply as *graphs*, and write $\mathcal{G}$ for the set of all graphs. Note that $\mathcal{G}$ is countably infinite. For a graph G, we write $V(G)$ and $E(G)$ for its vertex set and edge set, respectively, and $v(G) \stackrel{\text{def}}{=} |V(G)|$ and $e(G) \stackrel{\text{def}}{=} |E(G)|$ for their cardinalities. For $U \subseteq V(G)$, we write $G[U]$ for the subgraph of G induced by U, i.e., the graph with vertex set U and edge set $E(G) \cap \binom{U}{2}$. We use standard notation for common graphs. The complete graph on vertex set $[n]$ is denoted by K_n (so $E(K_n) = \binom{[n]}{2}$), the path on n vertices by P_n (i.e., $V(P_n) = [n]$ and $E(P_n) = \{\{i, i+1\} : i \in [n-1]\}$), and the edgeless graph with n vertices by I_n (i.e., $V(I_n) = [n]$ and $E(I_n) = \emptyset$).

2 Induced Subgraph Density

Let H and G be graphs with $v(H) \leq v(G)$. The central notion in this survey article is the *induced subgraph density* of H in G, or simply the *H-density* in G, defined by

$$p(H, G) \stackrel{\text{def}}{=} \mathbb{P}[G[\mathbf{U}] \simeq H],$$

where $\mathbf{U}$ is a uniformly random subset of $V(G)$ of size $v(H)$, and $\simeq$ denotes graph isomorphism. In words, $p(H, G)$ is the probability that a uniformly random set of $v(H)$ vertices of G induces a subgraph isomorphic to H. For instance, if $H = K_2$ and $G = K_3$, then $p(H, G) = 1$, since every induced subgraph of G on two vertices is isomorphic to H. As another example, if $H = K_2$ and $G = P_3$, then $p(H, G) = \frac{2}{3}$, since among the three induced subgraphs of G on two vertices, exactly two are isomorphic to H. By convention, we set $p(H, G) = 0$ if $v(H) > v(G)$.

Many asymptotic extremal problems can be phrased in terms of induced subgraph densities. A representative example is an asymptotic induced-subgraph version of a Turán-type problem: given a graph H and a finite set of forbidden graphs $\mathcal{F}$ such that there exists at least one graph avoiding every graph in $\mathcal{F}$, determine the maximum possible H-density in large graphs that avoid every graph in $\mathcal{F}$. Formally, the problem asks for the value of

$$T(H, \mathcal{F}) \stackrel{\text{def}}{=} \limsup_{n \to \infty} \max\{p(H, G) \,:\, v(G) = n \text{ and } p(F, G) = 0 \text{ for all } F \in \mathcal{F}\}.$$

When $H = K_2$ and $\mathcal{F} = \{K_3\}$, $T(H, \mathcal{F})$ is the asymptotic maximum edge density of triangle-free graphs, and Mantel's theorem states that $T(H, \mathcal{F}) = 1/2$ [14]. More

generally, when $H = K_2$ and $\mathcal{F} = \{K_{r+1}\}$ for an integer $r \geq 2$, Turán's theorem gives $T(H, \mathcal{F}) = 1 - \frac{1}{r}$ [21].

Another example is the Ramsey multiplicity problem, a counting analogue of the classical Ramsey number problem. Formally, for integers $s, t \geq 2$, the problem asks for the value of

$$M(s,t) \stackrel{\text{def}}{=} \liminf_{n\to\infty} \min\{p(K_s, G) + p(K_t, \overline{G}) \,:\, v(G) = n\},$$

where $\overline{G}$ is the complement of G (i.e., $V(\overline{G}) = V(G)$ and $E(\overline{G}) = \binom{V(G)}{2} \setminus E(G)$). Intuitively, G and $\overline{G}$ correspond to a 2-colouring of the edges of K_n (say, blue and red), and $p(K_s, G) + p(K_t, \overline{G})$ denotes the combined density of blue copies of K_s and red copies of K_t. It is known that $M(3,3) = \frac{1}{4}$ by Goodman's result [7], but the problem remains open for many other pairs (s,t). The flag algebra framework has been used to determine exact values in some cases (e.g., $M(3,4)$ and $M(3,5)$ [15]) and to obtain improved lower bounds in others (e.g., $M(4,4)$, $M(4,5)$, and $M(5,5)$ [10, 15]).

In both examples, the two arguments of the density function p play different roles: the second argument G is the graph under study, whereas the first argument H is part of the specification and denotes the pattern whose density is being measured. This convention is commonly used in extremal graph theory and is assumed in most applications of flag algebra. Accordingly, we call G the *host graph* and H the *pattern graph*. Typically, G is large and H is small, and $p(H, G)$ captures a local quantitative property of G.

Moreover, the density function p induces a representation of a graph G as an infinite-dimensional vector in $[0,1]^{\mathcal{G}}$ indexed by all graphs in $\mathcal{G}$. This representation is injective up to graph isomorphism: $p(\_, G) = p(\_, G')$ if and only if $G \simeq G'$. It also lets us use the product topology on $[0,1]^{\mathcal{G}}$ to study asymptotic properties of graph sequences, as we explain next.

3 Limiting Graphs

Let $(G_n)_{n\in\mathbb{N}}$ be a sequence of graphs. The sequence is *increasing* if $v(G_n) < v(G_{n+1})$ for all $n \in \mathbb{N}$. It is *convergent* if it is increasing and $\lim_{n\to\infty} p(H, G_n)$ exists for every graph H. Equivalently, the sequence of vector representations $(p(\_, G_n))_n$ mentioned at the end of the previous section converges pointwise in $[0,1]^{\mathcal{G}}$. Although this definition may appear restrictive—since it requires the limit to exist for every graph H—convergence is in fact common: every increasing sequence of graphs has a convergent subsequence. Indeed, the set $\mathcal{G}$ of graphs is countable[1], so the product space $[0,1]^{\mathcal{G}}$, equipped with the product topology, is

[1]Recall that by a graph we mean a finite simple graph G with $V(G) \subseteq \mathbb{N}$.

compact and metrizable. Hence it is sequentially compact, and the sequence of vectors $(p(\_, G_n))_{n\in\mathbb{N}}$ admits a convergent subsequence.

A function $\phi : \mathcal{G} \to [0,1]$ is a *limiting graph*[2] if there exists a convergent sequence of graphs $(G_n)_{n\in\mathbb{N}}$ such that

$$\phi(H) = \lim_{n\to\infty} p(H, G_n) \qquad \text{for every graph } H \in \mathcal{G}.$$

Let Φ denote the set of all limiting graphs. A useful mental picture (formalised, for example, via graphons [13]) is that a limiting graph represents a random infinite undirected graph with a continuum number of vertices, and $\phi(H)$ gives the average induced subgraph density of the pattern H in this infinite host graph. Every convergent sequence $(G_n)_n$ determines a unique limiting graph ϕ, but the same limiting graph may arise from multiple sequences.

Using limiting graphs, we can restate the extremal problems from Section 2 more cleanly.[3] The asymptotic induced-subgraph Turán-type problem can be formulated as the optimisation problem

$$T(H, \mathcal{F}) \stackrel{\text{def}}{=} \sup \{\phi(H) \,:\, \phi \in \Phi \text{ and } \phi(F) = 0 \text{ for all } F \in \mathcal{F}\}.$$

Similarly, since $p(K_t, \overline{G}) = p(I_t, G)$, the Ramsey multiplicity problem can be written as

$$M(s,t) \stackrel{\text{def}}{=} \inf \{\phi(K_s) + \phi(I_t) \,:\, \phi \in \Phi\}.$$

Here, I_t denotes the edgeless graph on t vertices.

A main message of this survey is that flag algebra provides a logical framework for reasoning about limiting graphs in Φ, and in particular for establishing bounds in asymptotic extremal problems. For example, it offers a syntax for expressing bounds such as

$$T(H, \mathcal{F}) \le c \qquad \text{and} \qquad M(s,t) \ge c'$$

for $c, c' \in \mathbb{R}$, together with a semantics parameterised by limiting graphs in Φ. In this way, such bounds become statements about limiting graphs. Flag algebra also comes with strategies for proving these bounds, some of which can be automated. In the next two sections, we present the syntax and semantics of this logic, and a popular proof strategy for it.

[2] In [17], such a function is called a *positive homomorphism*. We avoid this terminology because it can be opaque outside the algebraic setting, and we want to emphasise the interpretation of ϕ as a limit object representing an infinite graph.

[3] The correctness of this restatement follows from Corollary 3.4 of [17] and the fact that the asymptotic induced-subgraph Turán-type problem can be rephrased in terms of unconstrained optimisation.

4 Syntax and Semantics of Flag Algebra

The syntax of flag algebra consists of *density expressions* E and *assertions* A. Density expressions generalise pattern graphs H and denote real-valued functions on limiting graphs. Assertions express qualitative properties of limiting graphs using density expressions and logical connectives.

Formally, the syntax is defined as follows:

$$\begin{aligned} \text{Density Expressions} \quad & E ::= H \mid r \cdot E \mid 0 \mid E + E \mid 1 \mid E \cdot E, \\ \text{Assertions} \quad & A ::= \mathbf{false} \mid \mathbf{true} \mid E \geq E \mid \neg A \mid A \vee A, \end{aligned}$$

where H ranges over graphs and r over real numbers. We denote the set of all density expressions by Expr and that of assertions by Assn. Density expressions are built from pattern graphs and the constants 0 and 1 using scalar multiplication, addition, and multiplication. Assertions are constructed from the Boolean constants **false** and **true**, comparisons between density expressions, negation, and disjunction.

Negation and disjunction have their usual meanings from classical logic, and induce other connectives (e.g., conjunction and implication) in the standard way:

$$A \wedge A' \stackrel{\text{def}}{=} \neg(\neg A \vee \neg A') \qquad \text{and} \qquad (A \implies A') \stackrel{\text{def}}{=} \neg A \vee A'.$$

Similarly, other comparison operators between density expressions (such as $=$, $>$, $\leq$, and $<$) can be defined using $\geq$ and the logical connectives. For instance,

$$(E_1 = E_2) \stackrel{\text{def}}{=} (E_1 \geq E_2) \wedge (E_2 \geq E_1) \quad \text{and} \quad (E_1 < E_2) \stackrel{\text{def}}{=} \neg(E_1 \geq E_2).$$

Finally, we use the abbreviations $-E \stackrel{\text{def}}{=} (-1) \cdot E$ and $E_1 - E_2 \stackrel{\text{def}}{=} E_1 + (-E_2)$, and we identify real constants $r \in \mathbb{R}$ with the density expressions $r \cdot 1$.

The semantics of density expressions E and assertions A has the form

$$[\![E]\!] : \Phi \to \mathbb{R} \qquad \text{and} \qquad [\![A]\!] : \Phi \to \mathbb{B},$$

where $\mathbb{B} \stackrel{\text{def}}{=} \{\mathit{true}, \mathit{false}\}$. It is defined by induction on the structure of E and A using the ordered commutative algebra structure of $\mathbb{R}$ and the Boolean algebra structure of $\mathbb{B}$. For a limiting graph $\phi \in \Phi$,

$$\begin{aligned} [\![H]\!]\,\phi &\stackrel{\text{def}}{=} \phi(H), & [\![r \cdot E]\!]\,\phi &\stackrel{\text{def}}{=} r \cdot [\![E]\!]\,\phi, \\ [\![0]\!]\,\phi &\stackrel{\text{def}}{=} 0, & [\![E_1 + E_2]\!]\,\phi &\stackrel{\text{def}}{=} [\![E_1]\!]\,\phi + [\![E_2]\!]\,\phi, \\ [\![1]\!]\,\phi &\stackrel{\text{def}}{=} 1, & [\![E_1 \cdot E_2]\!]\,\phi &\stackrel{\text{def}}{=} [\![E_1]\!]\,\phi \cdot [\![E_2]\!]\,\phi, \end{aligned}$$

and

$$\begin{aligned}
[\![\mathbf{false}]\!]\,\phi &\stackrel{\text{def}}{=} \mathit{false}, \qquad [\![\mathbf{true}]\!]\,\phi \stackrel{\text{def}}{=} \mathit{true},\\
[\![E_1 \geq E_2]\!]\,\phi &\stackrel{\text{def}}{=} \begin{cases} \mathit{true} & \text{if } [\![E_1]\!]\,\phi \geq [\![E_2]\!]\,\phi,\\ \mathit{false} & \text{otherwise,}\end{cases}\\
[\![\neg A]\!]\,\phi &\stackrel{\text{def}}{=} \begin{cases} \mathit{true} & \text{if } [\![A]\!]\,\phi = \mathit{false},\\ \mathit{false} & \text{otherwise,}\end{cases}\\
[\![A_1 \vee A_2]\!]\,\phi &\stackrel{\text{def}}{=} \begin{cases} \mathit{true} & \text{if } [\![A_1]\!]\,\phi = \mathit{true} \text{ or } [\![A_2]\!]\,\phi = \mathit{true},\\ \mathit{false} & \text{otherwise.}\end{cases}
\end{aligned}$$

We say that an assertion A is *valid* if $[\![A]\!]\,\phi = \mathit{true}$ for every limiting graph $\phi \in \Phi$. Note that the syntax and semantics of flag algebra coincide with those of arithmetic expressions and formulas in the quantifier-free fragment of standard first-order logic over the reals. Graphs H in density expressions play the role of real-valued variables, density expressions are arithmetic expressions over these variables, and assertions are formulas over these arithmetic expressions. As a result, many standard reasoning principles from first-order logic apply to flag algebra directly. For instance, the following assertions hold for all limiting graphs $\phi \in \Phi$:

$$\begin{aligned}
&\Big(H_1 \cdot (H_3 + H_2) + (4 \cdot H_3) \cdot H_1\Big) = \Big(H_1 \cdot H_2 + 5 \cdot (H_1 \cdot H_3)\Big),\\
&\Big((H_1 + H_2 \geq 0) \wedge (0 \geq H_2 - H_3)\Big) \implies H_1 + H_3 \geq 0.
\end{aligned}$$

Lemma 4.1. *The rules of ordered commutative algebras are valid for density expressions in flag algebra. Also, the rules of Boolean algebras are valid assertions in flag algebra.*

What is special about flag algebra, compared with ordinary first-order logic over reals, is that the "variables" (i.e., graphs H) cannot be assigned arbitrary real values: their values must arise simultaneously from a single limiting graph ϕ, and are therefore correlated. One simple consequence is that two variables representing isomorphic graphs always have the same value.

Lemma 4.2. *If graphs H and H' are isomorphic, then the equation $H = H'$ is valid.*

Proof. Let ϕ be an arbitrary limiting graph. Then, there exists a converging sequence $(G_n)_{n\in\mathbb{N}}$ of graphs such that $\lim_{n\to\infty} p(F, G_n) = \phi(F)$ for all graphs $F \in \mathcal{G}$. Since H and H' are isomorphic, we have $p(H, G_n) = p(H', G_n)$ for all $n \in \mathbb{N}$. Therefore, $\phi(H) = \lim_{n\to\infty} p(H, G_n) = \lim_{n\to\infty} p(H', G_n) = \phi(H')$. Since ϕ was arbitrary, the equation $H = H'$ is valid. □

The next useful consequences follow from the connection between limiting graphs and probability theory. For each graph H and limiting graph ϕ, the value $\phi(H)$ intuitively means the probability that, if we choose a subset of $v(H)$ vertices uniformly at random in the infinite graph represented by ϕ, the induced subgraph on the chosen vertices is isomorphic to H. From this intuition, we see that $\phi(H) \in [0,1]$ and that the sum of $\phi(H')$ over all non-isomorphic graphs H' on a fixed vertex set $[n]$ equals 1. For a finite subset V of $\mathbb{N}$, we say that a set of graphs $\mathcal{H}_V$ is a *complete set of graphs on V* if every graph on the vertex set V is isomorphic to exactly one graph in $\mathcal{H}_V$. Using this terminology, we restate the above observations formally in the following lemmas.

Lemma 4.3. *The assertion $1 \geq H \wedge H \geq 0$ is valid for every graph H.*

Proof. Consider an arbitrary limiting graph ϕ. Then, there exists a converging sequence $(G_n)_{n\in\mathbb{N}}$ of graphs such that $\lim_{n\to\infty} p(F, G_n) = \phi(F)$ for all graphs $F \in \mathcal{G}$. Since $1 \geq p(H, G_n) \geq 0$ for all $n \in \mathbb{N}$, we have

$$\begin{aligned}
[\![1]\!]\,\phi = 1 \geq \lim_{n\to\infty} p(H, G_n) = \phi(H) = [\![H]\!]\,\phi, \text{ and} \\
[\![H]\!]\,\phi = \phi(H) = \lim_{n\to\infty} p(H, G_n) \geq 0 = [\![0]\!]\,\phi.
\end{aligned}$$

Since ϕ was arbitrary, the assertion $1 \geq H \wedge H \geq 0$ is valid. □

Lemma 4.4. *Let V be a finite subset of $\mathbb{N}$, and let $\mathcal{H}_V$ be a complete set of graphs on V. Then*

$$\sum_{H\in\mathcal{H}_V} H = 1$$

is valid.

Proof. Let ϕ be an arbitrary limiting graph. Then, there exists a converging sequence $(G_n)_{n\in\mathbb{N}}$ of graphs such that $\lim_{n\to\infty} p(F, G_n) = \phi(F)$ for all graphs $F \in \mathcal{G}$. Since $\mathcal{H}_V$ is a complete set of graphs on V, we have

$$\sum_{H\in\mathcal{H}_V} p(H, G_n) = 1$$

for all $n \in \mathbb{N}$. Therefore,

$$\left[\!\!\left[\sum_{H\in\mathcal{H}_V} H\right]\!\!\right] \phi = \sum_{H\in\mathcal{H}_V} \phi(H) = \sum_{H\in\mathcal{H}_V} \lim_{n\to\infty} p(H, G_n) = \lim_{n\to\infty} \sum_{H\in\mathcal{H}_V} p(H, G_n) = 1 = [\![1]\!]\,\phi.$$

Since ϕ was arbitrary, the equation $\sum_{H\in\mathcal{H}_V} H = 1$ is valid. □

We also note two further consequences that are heavily used in applications of flag algebra. They say that the density expression G for a graph G can be written as a linear combination of larger graphs, and that the multiplicative unit 1 and the multiplication in density expressions can be eliminated. In the lemmas below, let V be a finite subset of $\mathbb{N}$, and let $\mathcal{H}_V$ be a complete set of graphs on V.

Lemma 4.5. *For every graph H with $v(H) \leq |V|$,*

$$H = \sum_{H' \in \mathcal{H}_V} p(H, H') \cdot H'$$

is valid.

Proof. We use Lemma 2.2 from [17] in the proof.

Let ϕ be an arbitrary limiting graph. Then, there exists a converging sequence $(G_n)_{n\in\mathbb{N}}$ of graphs such that $\lim_{n\to\infty} p(F, G_n) = \phi(F)$ for all graphs $F \in \mathcal{G}$. Pick $n_0 \in \mathbb{N}$ such that $v(G_{n_0}) \geq |V|$. Then, since $\mathcal{H}_V$ is a complete set of graphs on V, by Lemma 2.2 from [17], we have

$$p(H, G_n) = \sum_{H' \in \mathcal{H}_V} p(H, H') \cdot p(H', G_n)$$

for all $n \geq n_0$. Therefore,

$$\begin{aligned}
[\![H]\!]\,\phi = \phi(H) &= \lim_{n\to\infty} p(H, G_n) \\
&= \lim_{n\to\infty} \sum_{H' \in \mathcal{H}_V} p(H, H') \cdot p(H', G_n) \\
&= \sum_{H' \in \mathcal{H}_V} p(H, H') \cdot \lim_{n\to\infty} p(H', G_n) \\
&= \sum_{H' \in \mathcal{H}_V} p(H, H') \cdot \phi(H') \\
&= \sum_{H' \in \mathcal{H}_V} p(H, H') \cdot [\![H']\!]\,\phi = \left[\!\!\left[\sum_{H' \in \mathcal{H}_V} p(H, H') \cdot H' \right]\!\!\right] \phi.
\end{aligned}$$

Since ϕ was arbitrary, the equation $H = \sum_{H' \in \mathcal{H}_V} p(H, H') \cdot H'$ is valid. □

Lemma 4.6. *If $\emptyset$ is the graph with the empty vertex set, then*

$$1 = \emptyset$$

is valid. Also, for every pair of graphs H_1 and H_2 with $v(H_1) + v(H_2) \leq |V|$, we have the valid equation

$$H_1 \cdot H_2 = \sum_{H \in \mathcal{H}_V} r_H \cdot H,$$

where

$$r_H \stackrel{\text{def}}{=} \mathbb{P}[H[\mathbf{U}_1] \simeq H_1 \text{ and } H[\mathbf{U}_2] \simeq H_2],$$

with $\mathbf{U}_1$ *and* $\mathbf{U}_2$ *being uniformly random disjoint subsets of* $V(H)$ *of sizes* $v(H_1)$ *and* $v(H_2)$*, respectively.*

Proof. We use Lemmas 2.2 and 2.3 from [17] in the second part of the proof.

Let ϕ be an arbitrary limiting graph, and let $(G_n)_{n\in\mathbb{N}}$ be a converging sequence of graphs such that $\lim_{n\to\infty} p(F, G_n) = \phi(F)$ for all graphs $F \in \mathcal{G}$.

Since $p(\emptyset, G_n) = 1$ for all $n \in \mathbb{N}$, we have

$$[\![1]\!]\,\phi = 1 = \lim_{n\to\infty} p(\emptyset, G_n) = \phi(\emptyset) = [\![\emptyset]\!]\,\phi.$$

The validity of $1 = \emptyset$ follows from the arbitrariness of ϕ.

Pick $n_0 \in \mathbb{N}$ such that $v(G_{n_0}) \geq |V|$. Then, for all $n \geq n_0$, applying the aforementioned lemmas from [17] yields

$$\left| p(H_1, G_n) \cdot p(H_2, G_n) - \sum_{H\in\mathcal{H}_V} r_H \cdot p(H, G_n) \right| \leq \frac{(v(H_1) + v(H_2))^C}{v(G_n)}$$

for some constant C. Since $v(G_n) \to \infty$ as $n \to \infty$, we have

$$\lim_{n\to\infty} (p(H_1, G_n) \cdot p(H_2, G_n)) = \lim_{n\to\infty} \sum_{H\in\mathcal{H}_V} r_H \cdot p(H, G_n).$$

Thus,

$$\begin{aligned}
[\![H_1 \cdot H_2]\!]\,\phi &= \phi(H_1) \cdot \phi(H_2) \\
&= \lim_{n\to\infty} p(H_1, G_n) \cdot \lim_{n\to\infty} p(H_2, G_n) \\
&= \lim_{n\to\infty} (p(H_1, G_n) \cdot p(H_2, G_n)) \\
&= \lim_{n\to\infty} \sum_{H\in\mathcal{H}_V} r_H \cdot p(H, G_n) \\
&= \sum_{H\in\mathcal{H}_V} r_H \cdot \lim_{n\to\infty} p(H, G_n) = \sum_{H\in\mathcal{H}_V} r_H \cdot \phi(H) = \left[\!\!\left[\sum_{H\in\mathcal{H}_V} r_H \cdot H \right]\!\!\right] \phi.
\end{aligned}$$

Since ϕ was arbitrary, the equation

$$H_1 \cdot H_2 = \sum_{H\in\mathcal{H}_V} r_H \cdot H$$

is valid, as claimed. □

We now illustrate the syntax and semantics of flag algebra by expressing the asymptotic extremal problems from Section 2. For a graph H and a finite set of forbidden graphs $\mathcal{F}$ such that there exists at least one graph avoiding every graph in $\mathcal{F}$, the Turán-type problem is to find the smallest $c \in \mathbb{R}$ such that the implication

$$\left(\bigwedge_{F\in\mathcal{F}} F = 0\right) \implies H \le c \tag{1}$$

is valid in flag algebra, i.e., it holds for all limiting graphs ϕ. Here, c is an abbreviation of the density expression $c \cdot 1$. The antecedent $\bigwedge_{F\in\mathcal{F}} F = 0$ expresses that ϕ avoids all forbidden graphs in $\mathcal{F}$, and the consequent $H \le c$ states that the density of H in ϕ is at most c.

For the Ramsey multiplicity problem, given integers $s, t \ge 2$, we seek the largest $c' \in \mathbb{R}$ such that the following inequality is valid in flag algebra:

$$K_s + I_t \ge c'. \tag{2}$$

Here, I_t denotes the edgeless graph with t vertices.

How does flag algebra help us to solve the problems of finding such optimal constants c and c' in Equations (1) and (2)? The answer lies in reasoning principles for deriving valid assertions in flag algebra. We give a glimpse of this aspect with two examples here.

Consider the Turán-type problem with $H = K_2$ and $\mathcal{F} = \{K_3\}$, which asks for the best upper bound on the asymptotic edge density of triangle-free graphs. In the language of flag algebra, this asks for the smallest c such that the implication

$$K_3 = 0 \implies K_2 \le c$$

is valid. By Mantel's theorem [14], the optimal constant is $c = 1/2$. We prove validity for $c = 1/2$.

Write  , , , and for the edgeless graph, the single-edge graph, the two-edge graph, and the complete graph K_3 on the vertex set [3], respectively.[4] We use the following two valid assertions:

$$K_2 = \Big(0 \cdot \text{} + \tfrac{1}{3} \cdot \text{} + \tfrac{2}{3} \cdot \text{} + 1 \cdot \text{}\Big), \tag{3}$$

$$\Big(\text{} - \tfrac{1}{3} \cdot \text{} - \tfrac{1}{3} \cdot \text{} + \text{}\Big) \ge 0. \tag{4}$$

The first is an instance of Theorem 4.5; the second can be proved using the reasoning principle explained in the next section. Assuming $K_3 = 0$, we derive the

[4] The exact correspondence between the dots in these notations and the labels $1, 2, 3$ is immaterial; any fixed choice yields isomorphic graphs.

desired bound as follows (each step holds for all $\phi \in \Phi$ with $\phi(K_3) = 0$):

$K_2 = 0 \cdot$  $+ \frac{1}{3} \cdot$  $+ \frac{2}{3} \cdot$  $+ 1 \cdot$  by Equation (3)

$\leq \left(0 \cdot\right.$  $+ \frac{1}{3} \cdot$  $+ \frac{2}{3} \cdot$  $\left. + 1 \cdot\right.$  $\right)$ by Equation (4) and Theorem 4.1

$+ \frac{1}{2} \cdot \left(\right.$  $- \frac{1}{3} \cdot$  $- \frac{1}{3} \cdot$  $+$  $\left.\right)$

$= \frac{1}{2} \cdot$  $+ \frac{1}{6} \cdot$  $+ \frac{1}{2} \cdot$  $+ \frac{3}{2} \cdot$  by Theorem 4.1

$= \frac{1}{2} \cdot$  $+ \frac{1}{6} \cdot$ $+ \frac{1}{2} \cdot$ $+ \frac{1}{2} \cdot$ since  $= K_3 = 0$

$\leq \frac{1}{2} \cdot$ $+ \frac{1}{2} \cdot$  $+ \frac{1}{2} \cdot$  $+ \frac{1}{2} \cdot$ by Theorems 4.1 and 4.3

$= \frac{1}{2} \cdot ($  $+$  $+$  $+$  $)$ by Theorem 4.1

$= \frac{1}{2} \cdot 1$ by Theorem 4.4

$= \frac{1}{2}.$

As a second example, consider the Ramsey multiplicity problem with $s = t = 3$, which asks for the largest c' such that

$$I_3 + K_3 \geq c'$$

is valid in flag algebra. By Goodman's result [7], the optimal constant is $c' = 1/4$. We prove validity for $c' = 1/4$ as follows:

$I_3 + K_3 = 1 \cdot$  $+ 0 \cdot$  $+ 0 \cdot$  $+ 1 \cdot$ 

$\geq (1 \cdot$ $+ 0 \cdot$  $+ 0 \cdot$  $+ 1 \cdot$  $)$ by Equation (4) and Theorem 4.1

$- \frac{3}{4} \cdot \left(\right.$ $- \frac{1}{3} \cdot$  $- \frac{1}{3} \cdot$  $+$ $\left.\right)$

$= \frac{1}{4} \cdot$ $+ \frac{1}{4} \cdot$ $+ \frac{1}{4} \cdot$ $+ \frac{1}{4} \cdot$ by Theorem 4.1

$= \frac{1}{4} \cdot ($ $+$ $+$ $+$ $)$ by Theorem 4.1

$= \frac{1}{4} \cdot 1$ by Theorem 4.4

$= \frac{1}{4}.$

Both examples use a step that has not yet been justified, namely the inequality in Equation (4). The next section explains a reasoning principle of flag algebra that allows one to derive such inequalities, as well as the downward operator, a key tool for applying this principle.

5 Deriving Inequalities with the Downward Operator

In this section, we explain a common proof strategy in flag algebra. It allows one to prove nontrivial inequalities such as Equation (4). The main tool is the

downward operator, which transfers inequalities proved in a labelled setting back to the original (unlabelled) setting.

At a high level, the strategy proceeds by introducing a family of *labelled* variants of flag algebra and, for each variant, a sound mechanism for transferring valid inequalities in that variant to valid inequalities in the original flag algebra. Each variant is chosen so that certain inequalities become easier to prove. One then proves such "easy" inequalities in an appropriate variant and transfers them to the unlabelled setting. In the derivation of Equation (4), for example, we use the variant in which one vertex is labelled.

To make this precise, we need some additional definitions. A *type* τ is a graph on the vertex set $[k]$ for some $k \in \mathbb{N}$; we call k the *size* of τ. A *τ-labelled graph* is a pair $G^\tau = (G, \theta)$ consisting of a graph G and an injective map $\theta : [k] \to V(G)$ such that θ is an *embedding* of τ into G, i.e., for all $i, j \in [k]$,

$$\{i, j\} \in E(\tau) \quad \text{if and only if} \quad \{\theta(i), \theta(j)\} \in E(G).$$

We call τ the *type* of G^τ, and write $|G^\tau|$ for the underlying unlabelled graph G. We also write $V(G^\tau)$, $E(G^\tau)$, $v(G^\tau)$, and $e(G^\tau)$ for $V(G)$, $E(G)$, $v(G)$, and $e(G)$, respectively. For a fixed type τ, we denote the set of all τ-labelled graphs by $\mathcal{G}^\tau$. Note that $\mathcal{G}^\tau$ is countably infinite, since every graph considered in this survey article has a finite vertex set contained in $\mathbb{N}$.

We now repeat the concepts from the previous sections for τ-labelled graphs. For τ-labelled graphs $H^\tau = (H, \iota)$ and $G^\tau = (G, \theta)$, a function $h : V(H) \to V(G)$ is a *τ-homomorphism* if it is a homomorphism from H to G and satisfies $h(\iota(i)) = \theta(i)$ for all $i \in [k]$. We say that H^τ and G^τ are *τ-isomorphic*, denoted $H^\tau \simeq_\tau G^\tau$, if there exists a bijective τ-homomorphism from H^τ to G^τ whose inverse is also a τ-homomorphism from G^τ to H^τ.

For τ-labelled graphs $H^\tau = (H, \iota)$ and $G^\tau = (G, \theta)$, the *induced τ-labelled subgraph density* of H^τ in G^τ (or simply the *H^τ-density* in G^τ) is defined by

$$p_\tau(H^\tau, G^\tau) \stackrel{\text{def}}{=} \mathbb{P}\Big[\big(G[\theta([k]) \cup \mathbf{U}],\, \theta\big) \simeq_\tau H^\tau\Big],$$

where $\mathbf{U}$ is a uniformly random subset of $V(G) \setminus \theta([k])$ of size $v(H^\tau) - k$. A sequence of τ-labelled graphs $(G_n^\tau)_{n\in\mathbb{N}}$ is *increasing* if $v(G_n^\tau) < v(G_{n+1}^\tau)$. It *converges* if it is increasing and $\lim_{n\to\infty} p_\tau(H^\tau, G_n^\tau)$ exists for every τ-labelled graph $H^\tau \in \mathcal{G}^\tau$. A function $\phi : \mathcal{G}^\tau \to [0, 1]$ is a *τ-labelled limiting graph* if there exists a convergent sequence $(G_n^\tau)_{n\in\mathbb{N}}$ such that $\phi(H^\tau) = \lim_{n\to\infty} p_\tau(H^\tau, G_n^\tau)$ for every $H^\tau \in \mathcal{G}^\tau$. As in the unlabelled case, ϕ can be viewed (informally) as a random infinite graph; however, now k distinguished vertices in this infinite graph are labelled $1, \ldots, k$, and these labels determine an embedding of τ into the infinite graph. We write Φ^τ for the set of all τ-labelled limiting graphs.

The τ-labelled variant of flag algebra is defined analogously to the original one, replacing graphs, induced subgraph densities, convergent graph sequences, and limiting graphs by τ-labelled graphs, induced τ-labelled subgraph densities, convergent sequences of τ-labelled graphs, and τ-labelled limiting graphs, respectively. Its syntax consists of density expressions and assertions built from τ-labelled graphs and interpreted over Φ^τ. For convenience, we recall the syntax:

$$\begin{aligned} \text{Density Expressions} \quad & E^\tau ::= H^\tau \mid r \cdot E^\tau \mid 0^\tau \mid E^\tau + E^\tau \mid 1^\tau \mid E^\tau \cdot E^\tau, \\ \text{Assertions} \quad & A^\tau ::= \mathbf{false} \mid \mathbf{true} \mid E^\tau \geq E^\tau \mid \neg A^\tau \mid A^\tau \vee A^\tau, \end{aligned}$$

where H^τ ranges over τ-labelled graphs and r ranges over real numbers. We write Expr^τ for the set of density expressions and Assn^τ for the set of assertions in this variant.

All of Theorems 4.1 to 4.6 also hold in the τ-labelled variant, with the obvious modifications (replacing graphs by τ-labelled graphs and induced subgraph densities by induced τ-labelled subgraph densities). We record one immediate consequence of Theorem 4.1 and the labelled version of Theorem 4.6 for later use.

Corollary 5.1. *Let τ be a type of size k. Then, for all expressions $E_1^\tau, \ldots, E_m^\tau$ in Expr^τ,*

$$\Big(\sum_{i \in [m]} E_i^\tau \cdot E_i^\tau \Big) \geq 0^\tau$$

is a valid assertion in the τ-labelled variant of flag algebra.

Lemma 5.2. *Let τ be a type of size k. Then the equation*

$$1^\tau = (\tau, \mathrm{id}_{[k]})$$

is valid in the τ-labelled variant of flag algebra. Also, let $V \subseteq \mathbb{N}$ be finite with $|V| \geq k$, and let $\mathcal{H}_V^\tau$ be a set of τ-labelled graphs on V such that every τ-labelled graph on V is τ-isomorphic to exactly one element of $\mathcal{H}_V^\tau$. Then, for every pair of τ-labelled graphs H_1^τ and H_2^τ with $v(H_1^\tau) + v(H_2^\tau) - k \leq |V|$, the following equation is valid:

$$H_1^\tau \cdot H_2^\tau = \sum_{(H,\theta) \in \mathcal{H}_V^\tau} r_{(H,\theta)} \cdot (H, \theta),$$

where

$$r_{(H,\theta)} \stackrel{\text{def}}{=} \mathbb{P}\big[H\big[\theta([k]) \cup \mathbf{U}_1\big] \simeq_\tau H_1^\tau \text{ and } H\big[\theta([k]) \cup \mathbf{U}_2\big] \simeq_\tau H_2^\tau\big],$$

with $\mathbf{U}_1$ and $\mathbf{U}_2$ being uniformly random disjoint subsets of $V(H) \setminus \theta([k])$ of sizes $v(H_1^\tau) - k$ and $v(H_2^\tau) - k$, respectively.

The transfer mechanism from the τ-labelled variant to the original flag algebra is the *downward operator*. We define it by a general recipe that relates the variant and the original flag algebras via what we call an *adjoint pair of functions*. We explain this recipe next and then instantiate it to obtain the downward operator.

Fix a type τ, and consider functions $\gamma : \mathcal{G} \to \mathrm{FinMea}(\mathcal{G}^\tau)$ and $\alpha : \mathcal{G}^\tau \to \mathrm{FinMea}(\mathcal{G})$. Here we regard $\mathcal{G}$ and $\mathcal{G}^\tau$ as measurable spaces equipped with the discrete σ-algebras, and write $\mathrm{FinMea}(\mathcal{G})$ and $\mathrm{FinMea}(\mathcal{G}^\tau)$ for the sets of finite measures on $\mathcal{G}$ and $\mathcal{G}^\tau$, respectively. Intuitively, for $G \in \mathcal{G}$, the measure $\gamma(G)$ defines an unnormalised probability distribution over $\mathcal{G}^\tau$, and for $H^\tau \in \mathcal{G}^\tau$, the measure $\alpha(H^\tau)$ defines an unnormalised probability distribution over $\mathcal{G}$. We say that (α, γ) is an *adjoint pair* if, for all $H^\tau \in \mathcal{G}^\tau$ and $G \in \mathcal{G}$,

$$p(\alpha(H^\tau), G) = p_\tau(H^\tau, \gamma(G)), \tag{5}$$

where

$$p(\alpha(H^\tau), G) \stackrel{\text{def}}{=} \int_{\mathcal{G}} p(H, G) \left(\alpha(H^\tau)(dH)\right),$$
$$p_\tau(H^\tau, \gamma(G)) \stackrel{\text{def}}{=} \int_{\mathcal{G}^\tau} p_\tau(H^\tau, G^\tau) \left(\gamma(G)(dG^\tau)\right).$$

Equation (5) is reminiscent of a Galois connection or a categorical adjunction.[5] In automated verification, Galois connections are used to relate different interpretations of a programming language or a logical system, and in programming languages, categorical adjunctions are used to relate semantic settings. Similarly, we use adjoint pairs to relate the original flag algebra and its τ-labelled variant.

Let (α, γ) be an adjoint pair for a type τ. We say that α has *finite support* if, for every $H^\tau \in \mathcal{G}^\tau$, the support

$$\mathrm{supp}(\alpha(H^\tau)) \stackrel{\text{def}}{=} \{H \,:\, \alpha(H^\tau)(\{H\}) > 0\}$$

is finite. This condition allows us to define the following map $\alpha^\dagger$ from multiplication-free and 1^τ-free expressions in Expr^τ to density expressions in the original flag algebra:

$$\alpha^\dagger(H^\tau) \stackrel{\text{def}}{=} \sum_{H \in \mathrm{supp}(\alpha(H^\tau))} \alpha(H^\tau)(\{H\}) \cdot H, \qquad \alpha^\dagger(r \cdot E^\tau) \stackrel{\text{def}}{=} r \cdot \alpha^\dagger(E^\tau),$$
$$\alpha^\dagger(0^\tau) \stackrel{\text{def}}{=} 0, \qquad \alpha^\dagger(E_1^\tau + E_2^\tau) \stackrel{\text{def}}{=} \alpha^\dagger(E_1^\tau) + \alpha^\dagger(E_2^\tau).$$

[5] Readers familiar with functional analysis may also recognise the similarity with adjoint linear operators.

We also say that γ is *extendible to limiting graphs* if there exists a function $\gamma_\infty : \Phi \to \mathrm{FinMea}(\Phi^\tau)$ such that, for every convergent sequence of graphs $(G_n)_{n\in\mathbb{N}}$ and every limiting graph $\phi \in \Phi$,

$$\Big(\forall H \in \mathcal{G},\ \lim_{n\to\infty} p(H, G_n) = \phi(H)\Big) \\ \implies \Big(\forall H^\tau \in \mathcal{G}^\tau,\ \lim_{n\to\infty} p_\tau(H^\tau, \gamma(G_n)) = \int_{\Phi^\tau} \phi^\tau(H^\tau)\,(\gamma_\infty(\phi)(d\phi^\tau))\Big).$$

Here, Φ^τ is equipped with the σ-algebra induced by the product topology on $[0,1]^{\mathcal{G}^\tau}$, making it a measurable space, and $\gamma_\infty(\phi)$ is a finite measure on Φ^τ.

The next lemma states that if α has finite support and γ is extendible to limiting graphs, then the adjoint pair (α, γ) yields a sound transfer principle.

Lemma 5.3. *Let (α, γ) be an adjoint pair for a type τ. Assume that α has finite support, and let $\alpha^\dagger$ be the induced extension of α to multiplication-free expressions that do not contain 1^τ. Also assume that γ is extendible to limiting graphs. Then, for every valid inequality $E^\tau \geq 0^\tau$ in the τ-labelled variant, where E^τ is multiplication-free and 1^τ-free, the inequality $\alpha^\dagger(E^\tau) \geq 0$ is valid in the original flag algebra.*

Proof. Let $\gamma_\infty : \Phi \to \mathrm{FinMea}(\Phi^\tau)$ be a witness of the extendibility of γ to limiting graphs. Consider an arbitrary limiting graph $\phi \in \Phi$. We need to show that $[\![\alpha^\dagger(E^\tau)]\!]\,\phi \geq 0$. Using structural induction on E^τ, we will show that

$$[\![\alpha^\dagger(E^\tau)]\!]\,\phi = \int_{\Phi^\tau} [\![E^\tau]\!]\,\phi^\tau\left(\gamma_\infty(\phi)(d\phi^\tau)\right). \tag{6}$$

The desired conclusion then follows from this equation because $E^\tau \geq 0^\tau$ is valid in the τ-labelled variant and so the integrand $[\![E^\tau]\!]\,\phi^\tau$ is non-negative for every $\phi^\tau \in \Phi^\tau$.

Assume that E^τ is a τ-labelled graph H^τ. By the definition of a limiting graph, there exists a convergent sequence of graphs $(G_n)_n$ such that

$$\phi(H) = \lim_{n\to\infty} p(H, G_n) \quad \text{for all } H \in \mathcal{G}. \tag{7}$$

Using this, we derive Equation (6) as follows:

$$
\begin{aligned}
&\left[\!\left[\alpha^\dagger(H^\tau)\right]\!\right]\phi \\
&= \left[\!\left[\sum_{H\in\mathrm{supp}(\alpha(H^\tau))} \alpha(H^\tau)(\{H\})\cdot H\right]\!\right]\phi && \text{by the definition of } \alpha^\dagger \\
&= \sum_{H\in\mathrm{supp}(\alpha(H^\tau))} \alpha(H^\tau)(\{H\})\cdot\phi(H) && \text{by the definition of } [\![-]\!] \\
&= \sum_{H\in\mathrm{supp}(\alpha(H^\tau))} \alpha(H^\tau)(\{H\})\cdot\lim_{n\to\infty} p(H,G_n) && \text{by Equation (7)} \\
&= \lim_{n\to\infty}\sum_{H\in\mathrm{supp}(\alpha(H^\tau))} \alpha(H^\tau)(\{H\})\cdot p(H,G_n) && \text{by the finiteness of } \mathrm{supp}(\alpha(H^\tau)) \\
&= \lim_{n\to\infty} p(\alpha(H^\tau),G_n) && \text{by the definition of } p(\alpha(H^\tau),G_n) \\
&= \lim_{n\to\infty} p_\tau(H^\tau,\gamma(G_n)) && \text{by the adjointness of } (\alpha,\gamma) \\
&= \int_{\Phi^\tau} \phi^\tau(H^\tau)\left(\gamma_\infty(\phi)(d\phi^\tau)\right) && \text{by the extendibility of } \gamma \\
&= \int_{\Phi^\tau} [\![H^\tau]\!]\,\phi^\tau\left(\gamma_\infty(\phi)(d\phi^\tau)\right) && \text{by the definition of } [\![-]\!]\,.
\end{aligned}
$$

Next consider the case that $E^\tau = 0^\tau$. Then,

$$
\left[\!\left[\alpha^\dagger(0^\tau)\right]\!\right]\phi = [\![0]\!]\,\phi = 0 = \int_{\Phi^\tau} 0\left(\gamma_\infty(\phi)(d\phi^\tau)\right) = \int_{\Phi^\tau} [\![0^\tau]\!]\,\phi^\tau\left(\gamma_\infty(\phi)(d\phi^\tau)\right).
$$

The remaining cases can be proved by the induction hypothesis and the linearity of integration. We first derive the target equation for $E^\tau = r\cdot F^\tau$:

$$
\begin{aligned}
\left[\!\left[\alpha^\dagger(r\cdot F^\tau)\right]\!\right]\phi &= \left[\!\left[r\cdot\alpha^\dagger(F^\tau)\right]\!\right]\phi && \text{by the definition of } \alpha^\dagger \\
&= r\cdot\left(\left[\!\left[\alpha^\dagger(F^\tau)\right]\!\right]\phi\right) && \text{by the definition of } [\![-]\!] \\
&= r\cdot\int_{\Phi^\tau} [\![F^\tau]\!]\,\phi^\tau\left(\gamma_\infty(\phi)(d\phi^\tau)\right) && \text{by the induction hypothesis} \\
&= \int_{\Phi^\tau} r\cdot\left([\![F^\tau]\!]\,\phi^\tau\right)\left(\gamma_\infty(\phi)(d\phi^\tau)\right) && \text{by the linearity of integration} \\
&= \int_{\Phi^\tau} [\![r\cdot F^\tau]\!]\,\phi^\tau\left(\gamma_\infty(\phi)(d\phi^\tau)\right) && \text{by the definition of } [\![-]\!]\,.
\end{aligned}
$$

Next, we handle the case that $E^\tau = E_1^\tau + E_2^\tau$:

$$\begin{aligned}
&\left[\!\left[\alpha^\dagger(E_1^\tau + E_2^\tau)\right]\!\right]\phi \\
&\quad= \left[\!\left[\alpha^\dagger(E_1^\tau) + \alpha^\dagger(E_2^\tau)\right]\!\right]\phi && \text{by the definition of } \alpha^\dagger \\
&\quad= \left[\!\left[\alpha^\dagger(E_1^\tau)\right]\!\right]\phi + \left[\!\left[\alpha^\dagger(E_2^\tau)\right]\!\right]\phi && \text{by the definition of } [\![-]\!] \\
&\quad= \int_{\Phi^\tau} [\![E_1^\tau]\!]\,\phi^\tau\left(\gamma_\infty(\phi)(d\phi^\tau)\right) && \text{by the induction hypothesis} \\
&\qquad+ \int_{\Phi^\tau} [\![E_2^\tau]\!]\,\phi^\tau\left(\gamma_\infty(\phi)(d\phi^\tau)\right) \\
&\quad= \int_{\Phi^\tau} \left([\![E_1^\tau]\!]\,\phi^\tau + [\![E_2^\tau]\!]\,\phi^\tau\right)\left(\gamma_\infty(\phi)(d\phi^\tau)\right) && \text{by the linearity of integration} \\
&\quad= \int_{\Phi^\tau} [\![E_1^\tau + E_2^\tau]\!]\,\phi^\tau\left(\gamma_\infty(\phi)(d\phi^\tau)\right) && \text{by the definition of } [\![-]\!]\,.
\end{aligned}$$

□

We are ready to define the downward operator for each type τ. Given an expression E^τ in Expr^τ, we first eliminate all multiplications and occurrences of the constant 1^τ in E^τ using Theorem 5.2, obtaining a multiplication-free, 1-free expression F^τ that is equal to E^τ in the τ-labelled variant of flag algebra. Although this elimination step is underspecified, it is typically performed from the innermost subexpression outward. The downward operator then maps F^τ to $\alpha_d^\dagger(F^\tau)$, where α_d is part of a specific adjoint pair (α_d, γ_d) for the type τ defined below. This adjoint pair satisfies the conditions in Theorem 5.3. Consequently, if $E^\tau \geq 0^\tau$ is a valid inequality in the τ-labelled variant, then $\alpha_d^\dagger(F^\tau) \geq 0$ is valid in the original flag algebra.

To define the adjoint pair (α_d, γ_d) mentioned above, consider a τ-labelled graph $H^\tau \in \mathcal{G}^\tau$ and an unlabelled graph $G \in \mathcal{G}$. For each $G^\tau \in \mathcal{G}^\tau$, let $\boldsymbol{\theta}_{G^\tau}$ be the uniformly random injection from $[k]$ to $V(G^\tau)$, and also let

$$q_{G^\tau} \stackrel{\text{def}}{=} \mathbb{P}\Big[\boldsymbol{\theta}_{G^\tau} \text{ is an embedding from } \tau \text{ to } |G^\tau| \text{ and } (|G^\tau|, \boldsymbol{\theta}_{G^\tau}) \simeq_\tau G^\tau\Big].$$

Note that $(\tau, \mathrm{id}_{[k]})$ is a τ-labelled graph whose underlying graph is τ itself, and that $q_{(\tau,\mathrm{id}_{[k]})} = |\mathrm{Aut}(\tau)|/v(\tau)!$ where $\mathrm{Aut}(\tau)$ is the set of all automorphisms on τ. Using this notation, we define finite measures $\alpha_d(H^\tau) \in \mathrm{FinMea}(\mathcal{G})$ and $\gamma_d(G) \in \mathrm{FinMea}(\mathcal{G}^\tau)$ as follows:

$$\alpha_d(H^\tau)(\{H\}) \stackrel{\text{def}}{=} \begin{cases} q_{H^\tau} & \text{if } |H^\tau| = H \\ 0 & \text{otherwise,} \end{cases}$$

$$\gamma_d(G)(\{G^\tau\}) \stackrel{\text{def}}{=} \begin{cases} \left(q_{(\tau,\mathrm{id})} \cdot p(\tau, G)\right) \div \left|\{G_0^\tau \in \mathcal{G}^\tau \;:\; |G_0^\tau| = G\}\right| & \text{if } |G^\tau| = G \\ 0 & \text{otherwise.} \end{cases}$$

Among the conditions in Theorem 5.3, it is straightforward to see that $\alpha_d(H^\tau)$ has finite support for every $H^\tau \in \mathcal{G}^\tau$. Also, it can be shown that γ_d is extendible to limiting graphs, although the proof is involved. See the proof of Theorem 3.5 in [17] for details. Finally, the following lemma establishes the adjointness of (α_d, γ_d).

Lemma 5.4. *The pair (α_d, γ_d) defined above is an adjoint pair for the type τ.*

Proof. We prove the lemma by substantially expanding the argument in the proof of Lemma 3.11 in [17] and adjusting the expanded version to our setting.

Fix an arbitrary τ-labelled graph $H^\tau \in \mathcal{G}^\tau$ and an arbitrary unlabelled graph $G \in \mathcal{G}$. We prove that

$$p(\alpha_d(H^\tau), G) = p_\tau(H^\tau, \gamma_d(G)).$$

We first simplify the left-hand side:

$$\begin{aligned} p(\alpha_d(H^\tau), G) &= \int_{\mathcal{G}} p(H, G)\,(\alpha_d(H^\tau)(dH)) \\ &= q_{H^\tau} \cdot p(|H^\tau|, G) \qquad \text{by the definition of } \alpha_d(H^\tau). \end{aligned}$$

Let

$$Z_G \stackrel{\text{def}}{=} \Big|\{G^\tau \in \mathcal{G}^\tau \,:\, |G^\tau| = G\}\Big|.$$

If $Z_G = 0$, then $p(\tau, G) = 0$ (there is no embedding of τ into G), hence $\gamma_d(G)$ is the zero measure and therefore $p_\tau(H^\tau, \gamma_d(G)) = 0$. Moreover, since H^τ contains τ on the labelled vertices, an induced copy of $|H^\tau|$ in G would in particular contain an induced copy of τ, contradicting $p(\tau, G) = 0$; hence $p(|H^\tau|, G) = 0$ and so the left-hand side is also 0. Thus, we may assume $Z_G > 0$.

Under this assumption, we simplify the right-hand side:

$$\begin{aligned} &p_\tau(H^\tau, \gamma_d(G)) \\ &\quad= \int_{\mathcal{G}^\tau} p_\tau(H^\tau, G^\tau)\,(\gamma_d(G)(dG^\tau)) \\ &\quad= \sum_{\substack{G^\tau \in \mathcal{G}^\tau \\ |G^\tau| = G}} p_\tau(H^\tau, G^\tau) \cdot \frac{q_{(\tau,\mathrm{id})} \cdot p(\tau, G)}{Z_G} \qquad \text{by the definition of } \gamma_d(G) \\ &\quad= (q_{(\tau,\mathrm{id})} \cdot p(\tau, G)) \cdot \sum_{\substack{G^\tau \in \mathcal{G}^\tau \\ |G^\tau| = G}} \frac{1}{Z_G}\, p_\tau(H^\tau, G^\tau). \end{aligned}$$

Therefore, it suffices to show that

$$q_{H^\tau} \cdot p(|H^\tau|, G) = (q_{(\tau,\mathrm{id})} \cdot p(\tau, G)) \cdot \sum_{\substack{G^\tau \in \mathcal{G}^\tau \\ |G^\tau| = G}} \frac{1}{Z_G}\, p_\tau(H^\tau, G^\tau). \tag{8}$$

Let $k = v(\tau)$, $m = v(H^\tau)$, and $n = v(G)$. Define $\boldsymbol{\theta}$ as a uniformly random injection from $[k]$ to $V(H^\tau)$, and $\boldsymbol{\theta}'$ as a uniformly random injection from $[k]$ to $V(G)$. Also, let $\mathbf{U}$ be a uniformly random subset of $V(G)$ of size m, and $\mathbf{U}'$ be a uniformly random subset of $V(G) \setminus \boldsymbol{\theta}'([k])$ of size $m - k$. Using these random variables, we define two events E and E' as follows:

- E is the event that $\boldsymbol{\theta}$ is an embedding of τ into $|H^\tau|$, the τ-labelled graph $(|H^\tau|, \boldsymbol{\theta})$ is τ-isomorphic to H^τ, and the induced subgraph $G[\mathbf{U}]$ is isomorphic to $|H^\tau|$.
- E' is the event that $\boldsymbol{\theta}'$ is an embedding of τ into G, and the τ-labelled graph $(G[\mathbf{U}' \cup \boldsymbol{\theta}'([k])], \boldsymbol{\theta}')$ is τ-isomorphic to H^τ.

The left-hand side of Equation (8) is equal to the probability of E. The random variables $\boldsymbol{\theta}$ and $\mathbf{U}$ are independent, and q_{H^τ} and $p(|H^\tau|, G)$ are the probabilities that $\boldsymbol{\theta}$ and $\mathbf{U}$ satisfy their respective conditions in the event E. Similarly, the right-hand side of Equation (8) is equal to the probability of E'. The factor $q_{(\tau,\mathrm{id})} \cdot p(\tau, G)$ in the right-hand side is the probability of the event E'' that $\boldsymbol{\theta}'$ is an embedding of τ into G, and the remaining factor equals the conditional probability $\mathbb{P}[E' \mid E'']$.

Thus, it suffices to show that $\mathbb{P}[E] = \mathbb{P}[E']$. We note that the joint distribution of $(\boldsymbol{\theta}, \mathbf{U})$ is the uniform distribution over the set of all pairs (θ, U) where θ is an injection from $[k]$ to $V(H^\tau)$ and U is a subset of $V(G)$ with size m. Similarly, the joint distribution of $(\boldsymbol{\theta}', \mathbf{U}')$ is the uniform distribution over the set of all pairs (θ', U') where θ' is an injection from $[k]$ to $V(G)$ and U' is a subset of $V(G) \setminus \theta'([k])$ with size $m - k$. Elementary counting confirms that these two sample spaces have the same cardinality $n!/((n-m)!(m-k)!)$. As a result, for such (U, θ) and (U', θ'), we have

$$\mathbb{P}[(\boldsymbol{\theta}, \mathbf{U}) = (\theta, U)] = \mathbb{P}[(\boldsymbol{\theta}', \mathbf{U}') = (\theta', U')] = \frac{(n-m)!(m-k)!}{n!}.$$

Therefore, to show that $\mathbb{P}[E] = \mathbb{P}[E']$, we just need to construct a bijection between the set of those (θ, U) satisfying E and the set of those (θ', U') satisfying E'. For each $U_0 \subseteq V(G)$ with size m such that $G[U_0]$ is isomorphic to $|H^\tau|$, we fix one isomorphism $f_{U_0} : V(H^\tau) \to U_0$ from $|H^\tau|$ to $G[U_0]$. Using this, we construct the desired bijection. Given a pair (θ, U) satisfying E, we define the corresponding pair (θ', U') satisfying E' as follows. By the definition of E, $G[U]$ is isomorphic to $|H^\tau|$. So, we have the isomorphism f_U from $|H^\tau|$ to $G[U]$. We define $\theta' : [k] \to V(G)$ by $\theta' \stackrel{\text{def}}{=} f_U \circ \theta$, and also define $U' \stackrel{\text{def}}{=} U \setminus \theta'([k])$. Then, (θ', U') satisfies the event E'. Furthermore, this construction is invertible: given a pair (θ', U') satisfying E', we can recover the corresponding pair (θ, U) satisfying E by letting $U \stackrel{\text{def}}{=} U' \cup \theta'([k])$ and $\theta \stackrel{\text{def}}{=} f_U^{-1} \circ \theta'$. This completes the proof of Equation (8) and thus of the lemma. □

We illustrate the downward operator by using it to derive Equation (4). Let τ be the unique type of size 1, i.e., the graph with a single vertex and no edges. A τ-labelled graph (G, θ) can be equivalently viewed as a graph G with one distinguished vertex (namely $\theta(1)$); we adopt this viewpoint below. By Theorem 5.1, the inequality

$$\left(\overset{\circ}{\bullet} - \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\right) \cdot \left(\overset{\circ}{\bullet} - \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\right) \geq 0^{\tau}$$

is valid in the τ-labelled variant of flag algebra. Here, $\overset{\circ}{\bullet}$ and $\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}$ are the τ-labelled edgeless graph and the τ-labelled single-edge graph on vertex set [2], respectively; the distinguished vertex is depicted as an empty circle.

We expand the product and then eliminate all multiplications using valid equalities:

$$\begin{aligned}
&\left(\overset{\circ}{\bullet} - \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\right) \cdot \left(\overset{\circ}{\bullet} - \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\right) \\
&\quad = \overset{\circ}{\bullet} \cdot \overset{\circ}{\bullet} - 2 \cdot \left(\overset{\circ}{\bullet} \cdot \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\right) + \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} \cdot \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} && \text{by the } \tau\text{-labelled version of Theorem 4.1} \\
&\quad = \left(\overset{\circ}{\bullet}\;\overset{\circ}{\bullet} + \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\right) - 2 \cdot \left(\tfrac{1}{2} \cdot \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} + \tfrac{1}{2} \cdot \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\;\overset{\circ}{\bullet}\right) && \text{by Theorem 5.2} \\
&\qquad\qquad + \left(\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} + \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\;\overset{\circ}{\bullet}\right) \\
&\quad = \overset{\circ}{\bullet}\;\overset{\circ}{\bullet} + \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} - \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} - \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\;\overset{\circ}{\bullet} + \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} + \overset{\circ}{\bullet}\;\overset{\circ}{\bullet}\;\overset{\circ}{\bullet} && \text{by the } \tau\text{-labelled version of Theorem 4.1.}
\end{aligned}$$

Here, $\overset{\circ}{\bullet}\;\overset{\circ}{\bullet}$, $\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}$, $\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}$, $\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\;\overset{\circ}{\bullet}$, $\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}$, and $\overset{\circ}{\bullet}\;\overset{\circ}{\bullet}\;\overset{\circ}{\bullet}$ are the six τ-labelled graphs on vertex set [3] with the obvious edge sets; as above, the distinguished vertex is shown as an empty circle.

Therefore, by Theorem 5.3,

$$\alpha_d^{\dagger}\left(\overset{\circ}{\bullet}\;\overset{\circ}{\bullet} + \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} - \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} - \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\;\overset{\circ}{\bullet} + \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} + \overset{\circ}{\bullet}\;\overset{\circ}{\bullet}\;\overset{\circ}{\bullet}\right) \geq 0$$

is a valid inequality in the original flag algebra. This inequality implies the one in Equation (4) because the left-hand side of the former is equal to that of the latter as shown by a series of valid equations below:

$$\begin{aligned}
&\alpha_d^{\dagger}\left(\overset{\circ}{\bullet}\;\overset{\circ}{\bullet} + \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} - \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} - \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\;\overset{\circ}{\bullet} + \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} + \overset{\circ}{\bullet}\;\overset{\circ}{\bullet}\;\overset{\circ}{\bullet}\right) \\
&\quad = \alpha_d\left(\overset{\circ}{\bullet}\;\overset{\circ}{\bullet}\right) + \alpha_d\left(\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\right) - \alpha_d\left(\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\right) - \alpha_d\left(\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\;\overset{\circ}{\bullet}\right) && \text{by the definition of } \alpha_d^{\dagger} \\
&\qquad\qquad + \alpha_d\left(\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\right) + \alpha_d\left(\overset{\circ}{\bullet}\;\overset{\circ}{\bullet}\;\overset{\circ}{\bullet}\right) \\
&\quad = \overset{\circ}{\bullet}\;\overset{\circ}{\bullet} + \tfrac{1}{3} \cdot \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} - \tfrac{2}{3} \cdot \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} - \tfrac{2}{3} \cdot \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\;\overset{\circ}{\bullet} + \tfrac{1}{3} \cdot \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} + \overset{\circ}{\bullet}\;\overset{\circ}{\bullet}\;\overset{\circ}{\bullet} && \text{by the definition of } \alpha_d \\
&\quad = \overset{\circ}{\bullet}\;\overset{\circ}{\bullet} - \tfrac{1}{3} \cdot \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} - \tfrac{1}{3} \cdot \overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet}\!\!-\!\!\overset{\circ}{\bullet} + \overset{\circ}{\bullet}\;\overset{\circ}{\bullet}\;\overset{\circ}{\bullet} && \text{by Theorem 4.1.}
\end{aligned}$$

6 Final Remarks

We have presented flag algebra from the perspective of computer scientists working in logic, programming languages, automated verification, and formal meth-

ods. Our presentation emphasises that flag algebra forms a logical system for establishing asymptotic inequalities in extremal graph theory, with a well-defined syntax, semantics, and proof strategies. We described in detail one common proof strategy: proving inequalities in a labelled variant of flag algebra—often starting from manifestly valid sum-of-squares inequalities—and then transferring them to the unlabelled setting using the downward operator. We also highlighted that this transfer mechanism relies on the notion of an adjoint pair, reminiscent of Galois connections and categorical adjunctions. Just as these notions relate different interpretations of programming languages or logics, adjoint pairs relate different variants of flag algebra.

This perspective suggests several research directions for logic and formal-methods researchers. First, it would be interesting to develop a proof theory for flag algebra. The strength and limitations of the downward-operator-based strategy have been analysed in the literature. For example, when the strategy is restricted to start from the sum-of-squares inequalities in Theorem 5.1, it is known to be incomplete [12, 4]. Similar analyses for other proof strategies in Razborov's original development [17]—such as those based on the upward operator or graph differentiation—would be valuable. More broadly, designing proof systems for flag algebra with good proof-theoretic properties while remaining effective in practice is an appealing direction.

Second, it would be interesting to develop more effective automation and software tools for constructing proofs in flag algebra and for solving the resulting optimisation problems. As noted in the introduction, Flagmatic has been highly successful for many extremal problems, but it largely relies on the downward-operator approach with sum-of-squares starting inequalities, and it does not scale well to medium-sized pattern graphs. Developing scalable automation that goes beyond the downward-operator/sum-of-squares paradigm is therefore an important challenge.

Acknowledgments

We would like to thank Joonkyung Lee, Sang-il Oum, Hyunwoo Lee, Taeyoung Kim, and Jaewon Moon for helping us to understand Razborov's flag algebra. This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korean Government(MSIT) (No. RS-2023-00279680).

References

[1] Rahil Baber and John M. Talbot. New Turán densities for 3-graphs. *Electron. J. Comb.*, 19(2):22, 2012.

[2] József Balogh, Ping Hu, Bernard Lidický, and Florian Pfender. Maximum density of induced 5-cycle is achieved by an iterated blow-up of 5-cycle. *Eur. J. Comb.*, 52:47–58, 2016.

[3] József Balogh, Ping Hu, Bernard Lidický, Florian Pfender, Jan Volec, and Michael Young. Rainbow triangles in three-colored graphs. *J. Comb. Theory B*, 126:83–113, 2017.

[4] Grigoriy Blekherman, Annie Raymond, Mohit Singh, and Rekha R Thomas. Simple graph density inequalities with no sum of squares proofs. *Combinatorica*, 40(4):455–471, 2020.

[5] Marcel K de Carli Silva, Fernando Mário de Oliveira Filho, and Cristiane Maria Sato. Flag algebras: a first glance. *Nieuw Arch. Wiskd.(5)*, 17(3):193–199, 2016.

[6] Victor Falgas-Ravry and Emil R. Vaughan. Applications of the semi-definite method to the Turán density problem for 3-graphs. *Comb. Probab. Comput.*, 22(1):21–54, 2013.

[7] A. W. Goodman. On sets of acquaintances and strangers at any party. *Amer. Math. Monthly*, 66(9):778–783, 1959.

[8] Andrzej Grzesik. On the maximum number of five-cycles in a triangle-free graph. *J. Comb. Theory B*, 102(5):1061–1066, 2012.

[9] Andrzej Grzesik. *Flag algebras in extremal graph theory*. PhD thesis, Jagiellonian University, 2015.

[10] Andrzej Grzesik, Joonkyung Lee, Bernard Lidickỳ, and Jan Volec. On tripartite common graphs. *Combinatorics, Probability and Computing*, 31(5):907–923, 2022.

[11] Hamed Hatami, Jan Hladký, Daniel Král', Serguei Norine, and Alexander A. Razborov. On the number of pentagons in triangle-free graphs. *J. Comb. Theory A*, 120(3):722–732, 2013.

[12] Hamed Hatami and Serguei Norine. Undecidability of linear inequalities in graph homomorphism densities. *Journal of the American Mathematical Society*, 24(2):547–565, 2011.

[13] László Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. American Mathematical Society, 2012. URL: `http://www.ams.org/bookstore-getitem/item=COLL-60`.

[14] Willem Mantel. Problem 28. *Wiskundige Opgaven*, 10:60–61, 1907.

[15] Olaf Parczyk, Sebastian Pokutta, Christoph Spiegel, and Tibor Szabó. New Ramsey multiplicity bounds and search heuristics. *Foundations of Computational Mathematics*, 25(5):1777–1814, 2025.

[16] Oleg Pikhurko, Jakub Sliacan, and Konstantinos Tyros. Strong forms of stability from flag algebra calculations. *J. Comb. Theory B*, 135:129–178, 2019.

[17] Alexander A. Razborov. Flag algebras. *J. Symb. Log.*, 72(4):1239–1282, 2007.

[18] Alexander A. Razborov. On the minimal density of triangles in graphs. *Comb. Probab. Comput.*, 17(4):603–618, 2008.

[19] Alexander A. Razborov. On 3-hypergraphs with forbidden 4-vertex configurations. *SIAM J. Discret. Math.*, 24(3):946–963, 2010.

[20] Alexander A Razborov. What is a flag algebra? *Notices of the AMS*, 60(10):1324–1327, 2013.

[21] Pál Turán. On an extremal problem in graph theory. *Matematikai és Fizikai Lapok*, 48:436–452, 1941.

THE CONCURRENCY COLUMN

by

Marino Miculan and Nobuko Yoshida

Department of Mathematics, Computer Science and Physics
University of Udine, Italy
marino.miculan@uniud.it

Department of Computer Science
University of Oxford, UK
nobuko.yoshida@cs.ox.ac.uk

The survey "Foundations of Runtime Monitoring through the Lens of Concurrency Theory" provides an overview of the theoretical advancements in runtime monitoring over the last decade. This work is of central importance to the theoretical computer science community as it addresses the fundamental challenge of monitorability, specifically characterizing which properties can be conclusively verified using finite execution traces while navigating the inherent loss of expressiveness compared to exhaustive methods like model checking.

A key aspect of the paper is the rigorous application of concurrency concepts to the monitoring domain. The authors utilize variations of Hennessy-Milner Logic with recursion (recHML) as a touchstone for specifications and employ process-algebraic languages, such as Milner's Calculus of Communicating Systems, to describe monitors. This approach allows for the use of structural operational semantics to formally define monitor behaviour and system interaction, enabling the creation of syntax-directed, correct-by-construction procedures for monitor synthesis. Therefore, the paper demonstrates how established concurrency concepts can address contemporary challenges in verifying complex distributed systems.

BEATCS no 148

FOUNDATIONS OF RUNTIME MONITORING THROUGH THE LENS OF CONCURRENCY THEORY

Luca Aceto
ICE-TCS, Department of Computer Science,
Reykjavik University, Iceland
Gran Sasso Science Institute, L'Aquila, Italy
luca@ru.is, luca.aceto@gssi.it

Antonis Achilleos
ICE-TCS, Department of Computer Science,
Reykjavik University, Iceland
antonios@ru.is

Elli Anastasiadi
Department of Computer Science, Aalborg University, Denmark
ellia@cs.aau.dk

Duncan Paul Attard
University of Malta, Msida, Malta
duncan.attard@um.edu.mt

Léo Exibard
LIGM, CNRS, Univ. Gustave Eiffel, Marne-la-Vallée, France
leo.exibard@univ-eiffel.fr

Adrian Francalanza
University of Malta, Msida, Malta
adrian.francalanza@um.edu.mt

Daniele Gorla
Department of Computer Science,
'Sapienza' University of Rome, Italy
gorla@di.uniroma1.it

Anna Ingólfsdóttir
ICE-TCS, Department of Computer Science,
Reykjavik University, Iceland
annai@ru.is

Karoliina Lehtinen
CNRS, Aix-Marseille University, LIS, Marseille, France
lehtinen@lis-lab.fr

Jana Wagemaker
Software Science Group, Radboud University, the Netherlands
jana.wagemaker@ru.nl

Abstract

This article surveys some of the work on the theoretical foundations of runtime monitoring carried out by various subsets of its authors over the last decade. It focuses on runtime monitoring of classic regular properties, of data-dependent properties and of hyperproperties. The paper also highlights the research philosophy guiding those studies, and the role that classic notions and techniques from concurrency theory have played in them.

1 Introduction

Runtime monitoring (also known as runtime verification) is a lightweight verification technique that can be used to determine whether a system satisfies some given requirements as it executes in its actual operating environment. The origins of modern runtime monitoring can be traced back to an influential workshop organised by Klaus Havelund and Grigore Rosu in 2001 [55], which then became the International Conference on Runtime Verification[1] that celebrated its 25th anniversary in 2025. To the best of our knowledge, Havelund and Rosu also coined the term 'runtime verification'.

[1] See https://runtime-verification.github.io/.

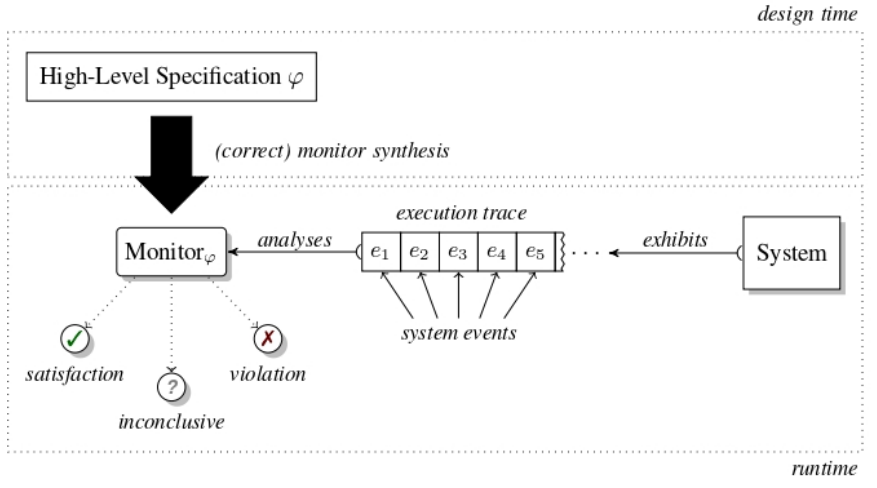


Figure 1: Runtime monitoring.

The general setting for (online) runtime monitoring is shown in Figure 1. Assume that we wish to determine whether a system operates according to some specification φ expressed in a formal specification language. In runtime monitoring, a computational entity called *monitor* is used to check whether the system under scrutiny satisfies φ or not. The task of a monitor for φ is to analyse the observable events in the current execution trace and to use the information it gleans from those observations to determine whether the system satisfies or violates the specification φ. Since the monitoring set-up should be part of the trusted computing base, it is desirable to synthesise monitors from specifications automatically and in a provably correct way. Indeed, as discussed in [1, 77] amongst other references, writing and maintaining monitors manually is costly and error prone.

Runtime monitoring is an increasingly popular verification technique that can be used in conjunction with classic approaches for the analysis and validation of software-based systems such as model checking [26, 42], deductive verification [22, 29, 64] and testing [69]. Indeed, several characteristics of runtime monitoring make it particularly suitable in the analysis of modern computing systems. For starters, runtime monitoring is a best-effort verification approach that focuses on analysing a *single system execution* (or possibly a few of them, as in offline monitoring or in the multiple-execution, online-monitoring setting studied in [20]). Thus, runtime monitoring does not suffer from the infamous state-explosion problem and is typically more scalable than variations on model checking, which are based on an exhaustive exploration of the state space of the system. (For a counterpoint,

see the complexity results presented in [66, 67].) Moreover, runtime monitoring typically treats the system as a black box and can be applied also when a model of the system under scrutiny or its source code are not available, for instance, because the system is proprietary. The black-box nature of runtime monitoring makes it an excellent fit for the analysis of systems that employ machine learning components and other subsystems based on artificial intelligence artefacts [36, 57, 76, 80]. Moreover, runtime monitoring can exploit the availability of multi-core systems, in that, while some of the processing units in a system carry out tasks pertaining to achieving system objectives, others can check whether a system execution satisfies the correctness requirements identified at design time. Finally, runtime monitoring takes place post-deployment while the system is running in its actual operating environment. This makes this approach very useful in settings, such as autonomous robotics and security, where it is practically impossible to imagine all the possible environments in which a system will operate [45, 75]. We are creative, but Nature is often adversarial and more creative than we are!

However, the aforementioned advantages of runtime monitoring as a verification technique come at the price of a loss of expressiveness. Indeed, as indicated in Figure 1, a monitor issues a verdict on whether the system under scrutiny satisfies the requirement expressed by φ after having observed a finite fragment of the system execution. There are properties for which this information is not sufficient to reach a conclusive verdict. As an example, consider the property 'the a action is performed infinitely often'. If a system can perform at least two different actions, then a monitor will never be able to determine conclusively whether that property is satisfied or violated, no matter what finite trace of actions it has observed thus far. Therefore, as Figure 1 makes clear, in general, the logic of monitors is (at least) three valued in that the verdict of a monitor for some property might remain forever inconclusive.

In light of the above discussion, a natural question to ask is whether one can characterise the collection of properties expressible in a given specification language that can be monitored in some given monitoring set-up, by what type of monitors and with what correctness guarantees. That question has been at the heart of the work on the theoretical foundations of monitorability carried out by various subsets of the authors of this article since our team's initial study of that topic in [50, 51]. Perhaps unsurprisingly, in light of our research background, we have attempted to answer it through the lens of concurrency theory. More specifically, our work on runtime monitoring has so far been based on, and guided by, the following design decisions.

- We have used variations on Hennessy-Milner logic with recursion [63] as our touchstone language for writing specifications of system behaviour. That logic is closely related to the modal μ-calculus [61] and can express the

operators in classic temporal logics [46]. Moreover, we use the standard branching- and linear-time semantics for the logics we study and do not change it to fit the runtime-monitoring setting. To our mind, doing so allows one to develop a theory of runtime monitoring for those logics that can be seamlessly combined, as needed, with other approaches to system verification such as model checking.

- We have employed variations on Milner's CCS [68] as our formalism for describing monitors. All our monitor-description languages come equipped with a structural operational semantics [72] that gives the semantics of monitors in terms of a labelled transition system [60]. We also use structural operational semantics to define the interaction between a system under scrutiny and a monitor observing its execution. Moreover, the operators that can be used to construct monitors are carefully chosen, in each case, to facilitate the syntax-directed definition of correct-by-construction, monitor-synthesis procedures from logical specifications. This allows us to reason about monitors, their behaviour and their correctness guarantees using classic proof techniques such as rule induction and structural induction.

- Our study of the specifications in our touchstone logics that can be monitored is relative to a given monitoring set-up. Indeed, the collection of monitorable properties depends on the power of the entities that carry out the monitoring task, viz. the monitors, and on the correctness guarantees one wants the monitoring process to uphold. Our tenet is that monitorability is best viewed as a spectrum of notions, which reflects the trade-off between what properties can be monitored and the correctness guarantees one would like to have—see the article [16], which relates our approach to classic notions of monitorability such as the one given by Pnueli and Zaks in [73] and those studied in [54].

- For each point in the spectrum of notions of monitorable properties, we have striven to identify fragments of our touchstone logic that are *expressively complete*, meaning that every monitorable property is logically equivalent to a formula in that fragment. Apart from their intrinsic theoretical interest, those characterisations may also have practical relevance. Indeed, the definition of automated, syntax-directed, monitor-synthesis procedures need only deal with formulae in the relevant fragment and can be optimised using its syntactic features. Moreover, designers specifying correctness properties can rest assured that if they specify system requirements using a formula in a given fragment, then (1) their requirements can be monitored at runtime with the correctness guarantees that accompany that fragment and (2) correct-by-construction monitors that do so can be generated automatically.

In the rest of this article, we will limit ourselves to presenting three exhibits describing the above-mentioned research approach at work. We will begin by discussing a classic setting in which system executions are expressed as (finite and) infinite traces (Section 2). We will then consider two variations on that setting dealing with monitoring of systems whose behaviour is data dependent (Section 3) and with centralised and decentralised monitoring of hyperproperties (Section 4). In each section, we will keep the presentation relatively informal, focusing on the main ideas and the key results that highlight the research philosophy underlying our work. We hope that readers will be enticed to check the scientific publications we cite for information on the technical details and on the software tools based on the theoretical results. The paper ends with some concluding remarks in Section 5.

Disclaimer This article is meant to be an appetiser offering a small taste of some of our work on runtime monitoring and highlighting its guiding principles, couched in classic concurrency theory. It is *not* a survey of the literature on runtime monitoring, which is vast and would deserve a handbook-length treatment. We refer our readers to [28, 65, 74, 78] and the references therein for overviews of the work done in the field of runtime monitoring.

2 Exhibit A: Adventures in classic monitorability

We start our journey by studying monitorability in a classic setting in which a system under scrutiny can perform observable actions from a non-empty, finite set Act, ranged over by a and b. Following Milner [68], we use τ as a distinguished unobservable system action that does not occur in Act and let $\mu \in \text{Act} \cup \{\tau\}$.

In this setting, infinite sequences of observable actions $t, u \in \text{Trc} = \text{Act}^{\omega}$, which we usually call *traces*, are a natural model for (the observable content of) system executions. Occasionally, we need to refer to *finite traces*, denoted by $s, r \in \text{Act}^{*}$, to represent objects such as a finite prefix of a system run. A trace and a finite trace with action a at their head are denoted by at and as, respectively. We write st for a trace that has prefix s and suffix t. A similar notation is used for prefixes and suffixes of finite traces. The empty finite trace ε is a prefix of every (finite) trace.

2.1 Syntax and semantics of recHML

As mentioned in the introduction, we use Hennessy-Milner logic with recursion recHML [19, 63] as our touchstone specification language. This is the collection

Syntax

$$
\begin{array}{rlll}
\varphi,\psi \in \textsc{rec}\text{HML} ::= & \mathsf{tt} & \text{(truth)} & \mid \mathsf{ff} \quad \text{(falsehood)} \\
\mid & \varphi \vee \psi & \text{(disjunction)} & \mid \varphi \wedge \psi \quad \text{(conjunction)} \\
\mid & \langle a \rangle \varphi & \text{(possibility)} & \mid [a]\varphi \quad \text{(necessity)} \\
\mid & \mathsf{min}\, X.\varphi & \text{(min. fixed point)} & \mid \mathsf{max}\, X.\varphi \quad \text{(max. fixed point)} \\
\mid & X & \text{(rec. variable)} &
\end{array}
$$

Linear-Time Semantics

$$
\begin{array}{rcll}
[\![\mathsf{tt}]\!]\rho & \stackrel{\text{def}}{=} & \textsc{Trc} & [\![\mathsf{ff}]\!]\rho \stackrel{\text{def}}{=} \emptyset \\
[\![\varphi_1 \wedge \varphi_2]\!]\rho & \stackrel{\text{def}}{=} & [\![\varphi_1]\!]\rho \cap [\![\varphi_2]\!]\rho & \\
[\![\varphi_1 \vee \varphi_2]\!]\rho & \stackrel{\text{def}}{=} & [\![\varphi_1]\!]\rho \cup [\![\varphi_2]\!]\rho & \\
[\![[a]\varphi]\!]\rho & \stackrel{\text{def}}{=} & \{t \mid \forall u \cdot t = au \text{ implies } u \in [\![\varphi]\!]\rho\} & \\
[\![\langle a \rangle \varphi]\!]\rho & \stackrel{\text{def}}{=} & \{t \mid \exists u \cdot t = au \text{ and } u \in [\![\varphi]\!]\rho\} & \\
[\![\mathsf{min} X.\varphi]\!]\rho & \stackrel{\text{def}}{=} & \bigcap \{T \mid [\![\varphi]\!]\rho[X \mapsto T] \subseteq T\} & \\
[\![\mathsf{max} X.\varphi]\!]\rho & \stackrel{\text{def}}{=} & \bigcup \{T \mid T \subseteq [\![\varphi]\!]\rho[X \mapsto T]\} & [\![X]\!]\rho \stackrel{\text{def}}{=} \rho(X)
\end{array}
$$

Figure 2: Syntax and linear-time semantics of recHML.

of formulae generated by the grammar in Figure 2. Such formulae are built from a countably infinite set of logical variables $X, Y \in \textsc{LVar}$, which are used to define recursive formulae expressing least or greatest fixed points. Formulae $\mathsf{min}\, X.\varphi$ and $\mathsf{max}\, X.\varphi$ bind free occurrences of the logical variable X in φ, inducing the usual notions of open/closed formulae and formula equality up to alpha-conversion. In what follows, we tacitly restrict ourselves to considering only closed formulae.

Apart from the standard Boolean constructs and the fixed-point operators, the logic is equipped with action-labelled possibility and necessity modalities from classic Hennessy-Milner logic [56].

The function $[\![-]\!]$ in Figure 2 gives the denotational semantics of recHML. It maps a formula φ to the set $[\![\varphi]\!]$ of the traces that satisfy φ, and gives the *linear-time* semantics of recHML. It uses valuations that map logical variables to sets of traces, $\rho : \textsc{LVar} \to \mathcal{P}(\textsc{Trc})$, to define the semantics by induction on the structure of the formulae. The notation $\rho[X \mapsto T]$ denotes the valuation that maps X to T and agrees with ρ on all the other logical variables. It is well known that the choice of the valuation ρ is immaterial for a closed formula φ. Therefore, we write $[\![\varphi]\!]$ for the set of traces that satisfy a closed formula φ.

Intuitively, $\rho(X)$ is the set of traces assumed to satisfy X. The cases for the Boolean operators are standard. An existential modal formula $\langle a \rangle \varphi$ denotes all

traces of the form at where t satisfies φ. A universal modal formula $[a]\varphi$ denotes all traces that do not start with a or have the form au for some u satisfying φ. The sets of traces satisfying the least and greatest fixed-point formulae, $\mathsf{min}\, X.\varphi$ and $\mathsf{max}\, X.\varphi$, are defined as the intersection (respectively, union) of all the pre-fixed points (respectively, post-fixed points) of the endofunction over the powerset of Trc induced by the formula φ. Intuitively, greatest fixed-point formulae may be used to describe safety properties—that is, properties that are satisfied by a trace t unless there is some finite prefix of t that provides evidence to the contrary. On the other hand, least fixed-point formulae express liveness properties—that is, properties that are satisfied by a trace t only if there is some finite prefix of t that witnesses that fact [75]. By way of example, as our readers might want to check, $[\![\mathsf{max}\, X.\langle a\rangle X]\!] = \{a^\omega\} = [\![\mathsf{max}\, X.(\langle a\rangle X \wedge X)]\!]$ whereas $[\![\mathsf{min}\, X.\langle a\rangle X]\!] = \emptyset$. Moreover, assuming that Act $= \{a, b\}$, the set $[\![\mathsf{min}\, X.(\langle a\rangle \mathsf{tt} \vee \langle b\rangle X)]\!]$ contains all traces apart from b^ω; thus, the formula $\mathsf{min}\, X.(\langle a\rangle \mathsf{tt} \vee \langle b\rangle X)$ expresses the liveness property 'the trace eventually contains an a'.

Two formulae φ and ψ in recHML are *logically equivalent* (over Trc) when $[\![\varphi]\!] = [\![\psi]\!]$.

Remark. It is well known that, even though it does not include a negation operator, the logic recHML is semantically closed under negation—that is, for every formula φ in recHML there is a formula $\overline{\varphi}$ in recHML such that $[\![\overline{\varphi}]\!] = \text{Trc} \setminus [\![\varphi]\!]$. The use of a logic without negation ensures that all formulae define monotonic endofunctions over $\mathcal{P}$(Trc), avoiding the need for syntactic restrictions over the collection of formulae.

A formula is *guarded* if every occurrence of each fixed-point variable appears within the scope of a modality within its fixed-point binding. For example, the formulae $\mathsf{max}\, X.\langle a\rangle X$ and $\mathsf{min}\, X.(\langle b\rangle \mathsf{tt} \vee \langle a\rangle X)$ are guarded, but $\mathsf{max}\, X.(\langle a\rangle X \wedge X)$ is not. Without loss of expressiveness, we assume that all formulae are guarded. Indeed, each formula is logically equivalent to a guarded one [35, 62].

Remark. The definitions of $[\![-]\!]$ and of logical equivalence apply equally well to other classic trace-based domains such as the collection of finite traces Act^* and that of finite and infinite traces $\text{Act}^* \cup \text{Act}^\omega$. The latter domain was dubbed the set of *finfinite traces* in [13].

Remark. Over the domain of infinite traces built from a finite set of actions Act, the modal operator $[a]$ can be expressed using $\langle a\rangle$ as follows:

$$[a]\varphi = \langle a\rangle\varphi \vee \bigvee_{b \in \text{Act}\setminus\{a\}} \langle b\rangle \mathsf{tt}.$$

However, this fails over finite or finfinite traces. For example, over those domains, there is no formula that does not use the necessity modal operators and is logically

equivalent to $[a]\mathsf{ff}$, which expresses the fact that a trace does not start with the action a. Indeed, as our readers can check, every formula that does not use the necessity modal operators and is satisfied by the empty trace is a tautology.

2.2 What are the monitorable recHML formulae?

Now that we have introduced our touchstone specification language recHML, we are ready to study which properties expressible in it are monitorable and with which correctness guarantees. However, in order to do so, we first need to define precisely when a formula in recHML is monitorable over Trc.

As we mentioned in the introduction, one of our tenets is that the notion of monitorability should be defined in terms of the computational devices that carry our the monitoring process, viz. the monitors. As depicted in Figure 1, a monitor may reach any one of *three* verdicts after analysing a finite trace: *acceptance*, which we denote by yes, *rejection*, which we write as no, and the *inconclusive* verdict, written end. Following [73], verdicts are *irrevocable*, meaning that a monitor cannot change its mind after it has issued a verdict.

At an abstract level, a monitor m can therefore be fully characterised by the set $A(m)$ of finite traces for which it issues an acceptance verdict yes and the set $R(m)$ of finite traces for which it issues a rejection verdict no. Since verdicts are irrevocable, the sets $A(m)$ and $R(m)$ are *extension closed*, that is, they satisfy the following natural closure properties:

- if $s \in A(m)$ then $sw \in A(m)$ for each $w \in \text{Act}^*$, and
- if $s \in R(m)$ then $sw \in R(m)$ for each $w \in \text{Act}^*$.

Another sanity criterion for monitors is that they be *consistent*, namely that $A(m)$ and $R(m)$ are disjoint. Consistent monitors cannot accept and reject the same finite trace.

For the moment, we assume that each monitor m is described by two sets of finite traces $A(m)$ and $R(m)$ satisfying the aforementioned closure properties. We will then see how those sets can be associated to monitors described using a process-algebraic language using their operational semantics, which details how monitors evolve when observing a trace describing the observable content of a system run.

Definition 2.1. A monitor m *accepts* a trace $t \in \text{Trc}$ iff $t = su$ for some $s \in A(m)$ and $u \in \text{Trc}$. Similarly, a monitor m *rejects* a trace $t \in \text{Trc}$ iff $t = su$ for some $s \in R(m)$ and $u \in \text{Trc}$.

Remark. The definition of when a monitor accepts or rejects a trace given in [13] is based on a binary operation, called *instrumentation*, whose structural operational

semantics describes how a monitor evolves when it observes a trace. That definition is equivalent to the one given in Definition 2.1. The same applies to the corresponding notions we use in Sections 3 and 4.

Here, we eschewed the introduction of different instrumentation operations to reduce the amount of technical machinery in the presentation of our results.

We are now ready to define the key concepts of monitor soundness and (partial) completeness with respect to a formula in recHML.

Definition 2.2 (Monitor Soundness and (Partial) Completeness over Traces)**.** Let φ be closed formula in recHML.

- A monitor m is *sound* for φ if for all $t \in \textsc{Trc}$:
  - if m accepts t then $t \in [\![\varphi]\!]$;
  - if m rejects t then $t \notin [\![\varphi]\!]$.
- A monitor m is *satisfaction-complete* for φ if for all $t \in \textsc{Trc}$, if $t \in [\![\varphi]\!]$ then m accepts t. It is *violation-complete* for φ if for all $t \in \textsc{Trc}$, if $t \notin [\![\varphi]\!]$ then m rejects t.
- A monitor m is *complete* for φ if it is both violation- and satisfaction-complete for it.

Soundness of a monitor for a formula is a non-negotiable requirement if monitors are to be used in a verification setting. Moreover, as our readers can easily check, it guarantees that a monitor is consistent. However, soundness alone is a very weak correctness guarantee; indeed, a monitor that neither accepts nor rejects any trace is sound for every property. The three flavours of completeness described above provide more monitor-correctness guarantees; they reflect the fact that runtime monitoring is, in general, less expressive than verification techniques such as model checking and, therefore, completeness might be unattainable using it for sufficiently powerful specification languages. Hence, to our mind, it is crucial to characterise the collection of properties that can be monitored in a sound and complete, violation- or satisfaction-complete way. Before presenting such characterisations, we introduce an operational model of monitors that we have employed to obtain those characterisation results.

The language we use to describe monitors, $m, n \in \textsc{Mon}$, is defined by the grammar and the transition rules in Figure 3. It is a variation on the restriction- and relabelling-free fragment of Milner's CCS [68], where we use verdicts $v \in \{\mathsf{yes}, \mathsf{no}, \mathsf{end}\}$ in lieu of the nil process and we endow monitors with conjunctive parallelism, $\otimes$, and disjunctive parallelism, $\oplus$. Intuitively, the $\otimes$ operator over

Syntax of monitors

$$m, n \in \textsc{Mon} ::= v \mid a.m \mid m + n \mid \mathsf{rec}\, x.m \mid x \mid m \otimes n \mid m \oplus n$$

Structural operational semantics of monitors

$$\textsc{mAct}\ \frac{}{a.m \xrightarrow{a} m} \qquad \textsc{mRec}\ \frac{}{\mathsf{rec}\, x.m \xrightarrow{\tau} m[\mathsf{rec}\, x.m/x]}$$

$$\textsc{mSelL}\ \frac{m \xrightarrow{\mu} m'}{m + n \xrightarrow{\mu} m'} \qquad \textsc{mSelR}\ \frac{n \xrightarrow{\mu} n'}{m + n \xrightarrow{\mu} n'} \qquad \textsc{mVer}\ \frac{}{v \xrightarrow{a} v}$$

$$\textsc{mPar}\ \frac{m \xrightarrow{a} m' \quad n \xrightarrow{a} n'}{m \odot n \xrightarrow{a} m' \odot n'} \qquad \textsc{mTauL}\ \frac{m \xrightarrow{\tau} m'}{m \odot n \xrightarrow{\tau} m' \odot n} \qquad \textsc{mVrE}\ \frac{}{\mathsf{end} \odot \mathsf{end} \xrightarrow{\tau} \mathsf{end}}$$

$$\textsc{mVrC1}\ \frac{}{\mathsf{yes} \otimes m \xrightarrow{\tau} m} \qquad \textsc{mVrC2}\ \frac{}{\mathsf{no} \otimes m \xrightarrow{\tau} \mathsf{no}}$$

$$\textsc{mVrD1}\ \frac{}{\mathsf{no} \oplus m \xrightarrow{\tau} m} \qquad \textsc{mVrD2}\ \frac{}{\mathsf{yes} \oplus m \xrightarrow{\tau} \mathsf{yes}}$$

Figure 3: Syntax and semantics of monitors.

monitors plays the role of conjunction over formulae in that a monitor of the form $m \otimes n$ can only reach the acceptance verdict yes if both m and n can do so. Similarly, the $\oplus$ operator over monitors is akin to disjunction over formulae in that a monitor of the form $m \oplus n$ can only reach the verdict yes if m or n can do so. Figure 3 presents the structural operational semantics of parallel monitors that formalises their behaviour. (We use the notation $\odot$ to stand for $\otimes$ or $\oplus$, that is, $\odot \in \{\otimes, \oplus\}$.) Rule mPar states that *both* submonitors need to be able to analyse an external action a for their parallel composition to transition with that action. The rules in Figure 3 also allow τ-transitions for the reconfiguration of parallel compositions of monitors. For instance, rules mVrC1 and mVrC2 describe the fact that, whereas yes verdicts are uninfluential in conjunctive parallel compositions, no verdicts supersede the verdicts of other monitors in conjunctive parallel compositions. (Figure 3 omits the symmetric rules.) The dual applies for yes and no verdicts in a disjunctive parallel composition, as described by rules mVrD1 and mVrD2. Rule mVrE applies to both forms of parallel composition and consolidates multiple

inconclusive verdicts. Rules mTauL and its dual mTauR (omitted) are contextual rules for these monitor reconfiguration steps. Finally, we highlight the transition rule for verdicts, describing the fact that, from a verdict state, any action can be analysed by transitioning to the same state; verdicts are thus *irrevocable*. The other rules are standard.

Remark. The conclusion of the transition rule for verdicts could be generalised to $v \xrightarrow{\mu} v$ as done in [6]. Doing so would make the algebraic theory of monitors slightly more elegant without changing any of the results we present in this section. For the sake of consistency with previous work, here we decided to stick to the rule given in [13, 50, 51].

Using the operational semantics of monitors, we can now define the sets of finite traces $A(m)$ and $R(m)$ associated with a monitor m. In what follows, we write $m \overset{s}{\Rightarrow} m'$ for some monitors m, m' and $s \in \text{Act}^*$ when there is a sequence of transitions leading from m to m' whose observable content is equal to s. For instance, $m \overset{\varepsilon}{\Rightarrow} m'$ means that m can transition to m' by performing a sequence of τ-labelled transitions and $m \overset{a}{\Rightarrow} m'$ holds when there are m_1 and m_2 such that $m \overset{\varepsilon}{\Rightarrow} m_1 \xrightarrow{a} m_2 \overset{\varepsilon}{\Rightarrow} m'$.

Definition 2.3. For each $m \in \text{Mon}$, we define

$$A(m) = \{s \mid m \overset{s}{\Rightarrow} \mathsf{yes}\} \quad \text{and} \quad R(m) = \{s \mid m \overset{s}{\Rightarrow} \mathsf{no}\}.$$

Example 2.1. As we observed earlier, $[\![\mathsf{max}\, X.\langle a\rangle X]\!] = \{a^\omega\}$ and therefore the formula $\mathsf{max}\, X.\langle a\rangle X$ expresses the fact that a system run only produces action a. Assume, for the sake of simplicity, that $\text{Act} = \{a, b\}$. Consider the monitor $m = \mathsf{rec}\, x.(a.x + b.\mathsf{no})$. It is not hard to see that $A(m)$ is empty and $R(m)$ contains all the strings in Act^* that have at least one occurrence of b. This means that m is sound and violation-complete for $\mathsf{max}\, X.\langle a\rangle X$. On the other hand, there is no monitor that is sound and satisfaction-complete for $\mathsf{max}\, X.\langle a\rangle X$. Indeed, any satisfaction-complete monitor n for that formula would have to accept the trace a^ω, since that trace satisfies $\mathsf{max}\, X.\langle a\rangle X$. This means that $a^k \in A(n)$ for some $k \geq 0$. Therefore, n would also accept the trace $a^k b^\omega$ and would be unsound. It follows that there is no sound and complete monitor for $\mathsf{max}\, X.\langle a\rangle X$.

Consider now the formula $\mathsf{min}\, X.(\langle a\rangle \mathsf{tt} \vee \langle b\rangle X)$. As we mentioned earlier in the paper, assuming that $\text{Act} = \{a, b\}$, that formula is satisfied by every trace apart from b^ω. Consider the monitor $m = \mathsf{rec}\, x.(a.\mathsf{yes} + b.X)$. It is not hard to see that $R(m)$ is empty and $A(m)$ contains all the strings in Act^* that have at least one occurrence of a. This means that m is sound and satisfaction-complete for $\mathsf{min}\, X.(\langle a\rangle \mathsf{tt} \vee \langle b\rangle X)$.

On the other hand, reasoning as above, our readers can convince themselves that there is no monitor that is sound and violation-complete for that formula.

As we shall see shortly, the non-existence of complete monitors for the formulae we have used in this example is an instance of a general phenomenon. Indeed, only formulae whose satisfaction can be determined by observing a bounded prefix of a trace can be monitored in a sound and complete way.

We are now ready to present the characterisation of the formulae in recHML that have sound and (partially) complete monitors. Below we write HML for the collection of recHML formulae that do not use fixed-point operators (that is, the formulae in classic Hennessy-Milner logic), maxHML for the set of recHML formulae that do not use the least-fixed-point operator and minHML for the fragment of recHML consisting of the formulae without occurrences of the greatest-fixed-point operator.

Theorem 2.1 (Aceto et al. [13])**.** *Let* $\varphi \in$ recHML.

1. *There is a sound and complete monitor* $m \in$ Mon *for* φ *iff* φ *is logically equivalent to a formula in* HML.
2. *There is a sound and violation-complete monitor* $m \in$ Mon *for* φ *iff* φ *is logically equivalent to a formula in* maxHML.
3. *There is a sound and satisfaction-complete monitor* $m \in$ Mon *for* φ *iff* φ *is logically equivalent to a formula in* minHML.

The proofs of the above-mentioned results are based on syntax-directed constructions that produce monitors with the stated correctness guarantees from formulae in the relevant fragments of recHML—see [13, Definitions 4.4 and 4.12]. For example, the following function constructs a sound and complete monitor from a formula φ in HML:

$$\mathsf{m}(\mathsf{ff}) \stackrel{\text{def}}{=} \mathsf{no} \quad \mathsf{m}(\varphi_1 \wedge \varphi_2) \stackrel{\text{def}}{=} \mathsf{m}(\varphi_1) \otimes \mathsf{m}(\varphi_2) \quad \mathsf{m}([a]\varphi) \stackrel{\text{def}}{=} a.\mathsf{m}(\varphi) + \sum_{b \neq a} b.\mathsf{yes}$$

$$\mathsf{m}(\mathsf{tt}) \stackrel{\text{def}}{=} \mathsf{yes} \quad \mathsf{m}(\varphi_1 \vee \varphi_2) \stackrel{\text{def}}{=} \mathsf{m}(\varphi_1) \oplus \mathsf{m}(\varphi_2) \quad \mathsf{m}(\langle a \rangle \varphi) \stackrel{\text{def}}{=} a.\mathsf{m}(\varphi) + \sum_{b \neq a} b.\mathsf{no}.$$

In the above-mentioned reference, we also show how to convert monitors into the formulae in a given fragment for which they are sound and (partially) complete [13, Definitions 4.7 and 4.13]. Moreover, we prove that monitors that make no use of the parallel conjunction and parallel disjunction operators suffice to establish Theorem 2.1 and that those monitors can be determinised—see [13, Proposition 3.11]. Intuitively, that result is based on the fact that a parallel monitor m of the type

we construct from formulae describes alternating automata recognising $A(m)$ and $R(m)$, which can then be converted into monitors that express NFAs and equivalent DFAs that accept $A(m)$ and $R(m)$. However, the price to be paid to carry out those conversions is hefty and unavoidable—see [13, Proposition 3.11] and [14].

Using non-constructive means, one can show a stronger version of Theorem 2.1(1) that applies to complete monitoring for any logic defined over traces. (See [13, Theorem 4.8].)

Theorem 2.2. *Let m be a monitor from a monitoring system where*

- *verdicts are irrevocable, that is, $A(m)$ and $R(m)$ are extension closed, and*
- *acceptance and rejections are defined as in Definition 2.1.*

For any property φ with a trace interpretation (not necessarily syntactically represented using recHML*), if m is sound and complete for φ then φ is logically equivalent to some formula in* HML*.*

Remark. The above-mentioned monitorability results hold when formulae are interpreted over *infinite* traces. If we consider finfinite traces instead, the monitorability picture changes considerably. Indeed, it turns out that, up to logical equivalence,

- only the formulae tt and ff have sound and complete monitors (see Lemma 5.4 in [13]);
- the only formulae that have sound and violation-complete monitors are those expressible in the safety fragment of recHML, which contains only the Boolean constants, conjunction, necessity modalities and greatest fixed-point operators (see [13, Proposition 5.8]); and
- the only formulae that have sound and satisfaction-complete monitors are those expressible in the co-safety fragment of recHML, which contains only the Boolean constants, disjunction, possibility modalities and least fixed-point operators (see [13, Proposition 5.8]).

We refer readers interested in reading more about our results dealing with classic monitorability to [10–12, 14–16, 49, 50, 50], which study the branching- and the linear-time settings and their relationships.

The monitoring tool detectEr for Erlang programs[2] is inspired by the theoretical developments presented in some of those references. However, that tool deals with properties of programs whose behaviour is data dependent, which go beyond the classic setting we have described so far. This realisation motivates the next step in our journey, where we study some of the theoretical developments that underpin the practice of runtime monitoring for data-driven systems [8, 27].

[2] See `https://duncanatt.github.io/detecter/` and the references [7, 24, 25, 37–39].

3 Exhibit B: Monitoring systems with data

In this section, we apply our research approach to a setting with data. We define recHML^d, an extension of recHML tailored to express properties of traces of system executions that contain data values. We also present some results pertaining to the characterisation of monitorable fragments of that logic from [8].

Since we consider linear-time properties as we did in Section 2, we model system executions as data *ω-words*. Typically, those are infinite words whose elements are pairs consisting of a letter from a finite alphabet and a *data value* from an infinite data domain. Since the finite alphabet plays no role in our developments and can be simulated [70], we omit it for simplicity. A data word is thus an infinite sequence of values from an infinite data domain.

For the rest of this section, we fix a countably infinite *data domain* $\mathbb{D}$, whose only predicate is '=' and is decidable. Concrete examples of such data domains include the sets $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\{0,1\}^*$ with equality, amongst others. An infinite (respectively, finite) *trace* is a data word, that is, an infinite (respectively, finite) sequence $t \in \mathbb{D}^\omega$ (respectively, $w \in \mathbb{D}^n$ for some $n \in \mathbb{N}$); the set of all infinite traces is denoted $\text{Trc}^d = \mathbb{D}^\omega$ (respectively, $\text{FTrc}^d = \mathbb{D}^*$ for finite traces).

3.1 Syntax and semantics of recHML^d

To express properties of traces of data values, we use an extension of recHML, called recHML^d. Its syntax and semantics are described in Figure 4. The new ingredients in the syntax of recHML^d are as follows. In addition to logical variables, formulae are built from a countably infinite set of data variables, $x, y \in \text{DVar}$, ranging over an infinite domain of data values, $d \in \mathbb{D}$. To reason about the data carried by process actions, modalities are augmented with decidable, quantifier-free Boolean *constraint expressions*, $b, c \in \text{BExp}$, defined over $\mathbb{D}$ and $\text{DVar} \cup \{\star\}$, where $\star \notin \text{DVar}$ is a placeholder variable for the current action $d \in \mathbb{D}$ produced by a system run. The free data variables $x \in \text{DVar}$ that appear in b are bound by existential and universal quantification constructs $\exists\, \boldsymbol{x}.\varphi$ and $\forall\, \boldsymbol{x}.\varphi$. Intuitively, the formula $\exists\, \boldsymbol{x}.\varphi$ is satisfied by a trace t if there is some $d \in \mathbb{D}$ such that t satisfies φ when the value of x is set to d. On the other hand, a trace t satisfies $\forall\, \boldsymbol{x}.\varphi$ when, for each $d \in \mathbb{D}$, t satisfies φ when the value of x is set to d.

The formal definition of the semantics of recHML^d requires several ingredients and is in the spirit of the one presented in [52] for a similar logic. We introduce them next to make the presentation self-contained. Readers who are not interested in the formal details of the semantics can gain an intuitive understanding of the constructs of recHML^d and of their expressiveness by reading the formulae in

RECHML^d Syntax

$$\varphi, \psi \in \text{RECHML}^d ::= \mathsf{tt} \mid \mathsf{ff} \mid \langle b \rangle \varphi \mid [b]\varphi \mid \exists \boldsymbol{x}.\varphi \mid \forall \boldsymbol{x}.\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \min X.(\varphi) \mid \max X.(\varphi) \mid X$$

Fragments

$$\varphi, \psi \in \text{CHML}^d ::= \mathsf{tt} \mid \langle b \rangle \varphi \mid \exists \boldsymbol{x}.\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \min X.(\varphi) \mid X$$

$$\varphi, \psi \in \text{SHML}^d ::= \mathsf{ff} \mid [b]\varphi \mid \forall \boldsymbol{x}.\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \max X.(\varphi) \mid X$$

$$\varphi, \psi \in \text{DISJHML}^d ::= \mathsf{tt} \mid \langle b \rangle \varphi \mid \exists \boldsymbol{x}.\varphi \mid \varphi \vee \psi \mid \min X.(\varphi) \mid X$$

$$\varphi, \psi \in \text{HML}^d ::= \mathsf{tt} \mid \mathsf{ff} \mid \langle b \rangle \varphi \mid [b]\varphi \mid \exists \boldsymbol{x}.\varphi \mid \forall \boldsymbol{x}.\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi$$

Semantics

$$[\![\,\mathsf{tt}\,]\!]^\rho_\delta \stackrel{\text{def}}{=} \text{TRC}^d \qquad [\![\,\mathsf{ff}\,]\!]^\rho_\delta \stackrel{\text{def}}{=} \varnothing \qquad [\![\,X\,]\!]^\rho_\delta \stackrel{\text{def}}{=} (\rho(X))(\delta)$$

$$[\![\,\langle b \rangle \varphi\,]\!]^\rho_\delta \stackrel{\text{def}}{=} \{t \mid (\exists u, d.t = du \text{ and } b\delta[\star \mapsto d] \Downarrow \mathsf{true} \text{ and } u \in [\![\,\varphi\,]\!]^\rho_\delta)\}$$

$$[\![\,[b]\,\varphi\,]\!]^\rho_\delta \stackrel{\text{def}}{=} \{t \mid (\forall u, d.(t = du \text{ and } b\delta[\star \mapsto d] \Downarrow \mathsf{true}) \text{ implies } u \in [\![\,\varphi\,]\!]^\rho_\delta)\}$$

$$[\![\,\exists \boldsymbol{x}.\varphi\,]\!]^\rho_\delta \stackrel{\text{def}}{=} \bigcup_{d \in \mathbb{D}} [\![\,\varphi\,]\!]^\rho_{\delta[x \mapsto d]}$$

$$[\![\,\forall \boldsymbol{x}.\varphi\,]\!]^\rho_\delta \stackrel{\text{def}}{=} \bigcap_{d \in \mathbb{D}} [\![\,\varphi\,]\!]^\rho_{\delta[x \mapsto d]}$$

$$[\![\,\varphi \vee \psi\,]\!]^\rho_\delta \stackrel{\text{def}}{=} [\![\,\varphi\,]\!]^\rho_\delta \cup [\![\,\psi\,]\!]^\rho_\delta$$

$$[\![\,\varphi \wedge \psi\,]\!]^\rho_\delta \stackrel{\text{def}}{=} [\![\,\varphi\,]\!]^\rho_\delta \cap [\![\,\psi\,]\!]^\rho_\delta$$

$$[\![\,\min X.(\varphi)\,]\!]^\rho_\delta \stackrel{\text{def}}{=} (\bigsqcap \{F \mid \lambda\delta'.[\![\,\varphi\,]\!]^{\rho[X \mapsto F]}_{\delta'} \sqsubseteq F\})(\delta)$$

$$[\![\,\max X.(\varphi)\,]\!]^\rho_\delta \stackrel{\text{def}}{=} (\bigsqcup \{F \mid F \sqsubseteq \lambda\delta'.[\![\,\varphi\,]\!]^{\rho[X \mapsto F]}_{\delta'}\})(\delta)$$

Expressions

$$b, c \in \text{BEXP} ::= \mathsf{true} \mid e = f \mid \neg b \mid b \wedge c \qquad e, f \in \text{EXP} ::= x \in \text{DVAR} \mid \star$$

Figure 4: Syntax and linear-time semantics of RECHML^d.

Example 3.1 and the natural-language descriptions of the properties they describe given there. The intuition provided there suffices to grasp the gist of the main results we present in this section.

We define the domains $\text{DEnv} = \text{DVar} \rightharpoonup \mathbb{D}$ of data environments, $\text{DInt} = \text{DEnv} \rightharpoonup 2^{\text{Trc}^d}$ of data interpretations, and $\text{TEnv} = \text{LVar} \rightharpoonup \text{DInt}$ of trace environments (where $A \rightharpoonup B$ denotes the set of partial functions from set A to set B). A *data environment*, $\delta \in \text{DEnv}$, is a partial function with a finite domain mapping data variables to values from $\mathbb{D}$; analogously, a *trace environment*, $\rho \in \text{TEnv}$, maps logical variables to data interpretations $F, G \in \text{DInt}$, that given δ, return a set of traces, whose intended meaning is the interpretation of the logical variable in the data environment δ.

The *linear-time* semantics of recHML^d is given by the denotational semantic function $[\![-]\!]$ defined inductively in Figure 4. Formulae are interpreted with respect to a trace environment ρ that gives meaning to logical variables, and a data environment δ that assigns values to data variables in Boolean constraint expressions. We write $[\![\varphi]\!]^{\rho}_{\delta}$ for the set of traces that satisfy φ with respect to ρ and δ.

An expression b defines a set of *external* system actions with respect to a data environment δ. An action d is in this set when the data value it carries satisfies b modulo δ, written $b\delta \Downarrow \mathsf{true}$. Possibility formulae $\langle\, b \,\rangle \varphi$ denote all the traces $t = du$ that begin with an action d that is in the action set described by $b\delta$ *and* whose tail u satisfies the continuation formula φ. Dually, necessity formulae $[\, b \,] \varphi$ describe all the traces that, *whenever* they begin with an action d that is in the action set described by $b\delta$, continue with a trace that satisfies φ. Note that, in the linear-time setting, necessity can be expressed as possibility: $[\, b \,] \varphi \equiv \langle\, \neg b \,\rangle \mathsf{tt} \vee \langle\, b \,\rangle \varphi$, and dually $\langle\, b \,\rangle \varphi = [\, \neg b \,] \mathsf{ff} \wedge [\, b \,] \varphi$. The existential quantifier $\exists \boldsymbol{x}.\varphi$ is interpreted as the set of traces that satisfy φ by assigning *some* $d \in \mathbb{D}$ to x; the universal quantifier $\forall\, \boldsymbol{x}.\varphi$ is the set of traces satisfying φ under *all* such assignments. Formulae are only interpreted with respect to data environments whose domain includes the set of free data variables occurring in them.

Since the logic does not have an explicit negation operator, for all φ the semantic function $[\![\varphi]\!]^{\rho}_{\delta}$ is monotonic in ρ over the complete lattice $(\text{DInt}, \sqsubseteq)$, where the partial order $\sqsubseteq$ corresponds to graph inclusion. Formally, it is defined, for all $F, G \in \text{DInt}$, as $F \sqsubseteq G$ whenever $\forall \delta \in \text{DEnv}.F(\delta) \subseteq G(\delta)$. As is standard in the modal μ-calculus, recursion is interpreted through fixed points: by the Knaster-Tarski theorem [79], $\mathsf{min}\, X.(\varphi)$ and $\mathsf{max}\, X.(\varphi)$ respectively correspond to the least and greatest fixed point of the operator that maps a data interpretation $F : \text{DEnv} \to 2^{\text{Trc}^d}$ to the data interpretation $\delta \mapsto [\![\varphi]\!]^{\rho[X \mapsto F]}_{\delta}$. This is the analogue of the operator used to define the semantics of recHML over traces in Section 2, lifted to the case of infinite alphabets by parameterising the interpretation by a data environment, in the spirit of [52]. To obtain the sought interpretation for $\mathsf{min}\, X.(\varphi)$

and $\mathsf{max}\, X.(\varphi)$, one then applies the least (respectively, greatest) fixed point of this operator (which is a function from data environments to sets of traces) to the current data environment δ.

Example 3.1. To give an intuition of the logic and its expressiveness, we present a few elementary recHML^d properties, along with their respective fragments, and discuss their meaning informally.

- The following formula in HML^d states that the first and the second data values in a trace are equal:

$$\varphi_{\text{f=s}} \stackrel{\text{def}}{=} \exists\, \boldsymbol{x}.\langle\, x = \star \,\rangle \langle\, x = \star \,\rangle \mathsf{tt}. \tag{1}$$

  Indeed, the only way for the first modality $\langle\, x = \star \,\rangle$ to be satisfied is if x takes the value of the first data value. Then, the second modality $\langle\, x = \star \,\rangle$ is satisfied exactly when the second value is equal to x, and hence to the first value.

- The following formula in cHML^d expresses the requirement that the first data value appears again in a trace:

$$\varphi_{\text{leak}} \stackrel{\text{def}}{=} \exists\, \boldsymbol{x}.\langle\, x = \star \,\rangle \mathsf{min}\, X.\, (\langle\, x = \star \,\rangle \mathsf{tt} \vee \langle\, x \neq \star \,\rangle X), \tag{2}$$

  where we use $x \neq \star$ to abbreviate $\neg(x = \star)$. As above, x stores the first data value. Then, we use recursion to look for another occurrence of that value. Intuitively, upon encountering a fixed-point variable X the formula recurses, that is, we unfold the formula by replacing X with the whole $\mathsf{min}\, X.(\varphi)$ that encloses it. Here, the formula recurses while it encounters values satisfying $x \neq \star$, and is satisfied (reaching tt) if it encounters a value satisfying $x = \star$, viz. the first value in the trace. Since this is a least-fixed-point formula (denoted by the use of min), the formula is satisfied only if it recurses finitely many times, which happens exactly when the first value appears again in a trace.

- The following formula in disjHML^d expresses the requirement that *some* data value appears at least twice in a trace:

$$\begin{aligned} \varphi_3 &\stackrel{\text{def}}{=} \exists\, \boldsymbol{x}.\varphi(x) \text{ where} \\ \varphi(x) &\stackrel{\text{def}}{=} \mathsf{min}\, X.\, (\langle\, x = \star \,\rangle \psi(x) \vee \langle\, x \neq \star \,\rangle X) \\ \psi(x) &\stackrel{\text{def}}{=} \mathsf{min}\, Y.\, (\langle\, x = \star \,\rangle \mathsf{tt} \vee \langle\, x \neq \star \,\rangle Y) \end{aligned} \tag{3}$$

  For a given value of x, the formula is satisfied only if this value is found once (first disjunct of $\varphi(x)$) and then again (first disjunct of $\psi(x)$). Overall, the formula is satisfied whenever there exists such a value, which thus appears twice.

- The following formula states that all data values appearing in a trace are pairwise distinct (negation by dualisation of the DISJHML^d formula above):

$$\begin{aligned} \varphi_4 &\stackrel{\text{def}}{=} \forall\, \boldsymbol{x}.\varphi(x) \text{ where} \\ \varphi(x) &\stackrel{\text{def}}{=} \mathsf{max}\, X.\, ([\, x = \star\,]\, \psi(x) \wedge [\, x \neq \star\,]\, X) \\ \psi(x) &\stackrel{\text{def}}{=} \mathsf{max}\, Y.\, ([\, x = \star\,]\, \mathsf{ff} \wedge [\, x \neq \star\,]\, Y) \end{aligned} \tag{4}$$

  Dually to the one we just discussed, this formula is *not* satisfied whenever some value appears twice in a trace.

- The following formula in RECHML^d states that there exists a data value that never appears:

$$\varphi_5 \stackrel{\text{def}}{=} \exists\, \boldsymbol{x}.\mathsf{max}\, X.\, ([\, x = \star\,]\, \mathsf{ff} \wedge [\, x \neq \star\,]\, X) \tag{5}$$

  As for φ_4, the greatest-fixed-point operator max allows one to forbid a data value (existentially guessed using the $\exists$ quantifier) from appearing in a trace. Indeed, once we have guessed a value for x, the formula

$$\mathsf{max}\, X.\, ([\, x = \star\,]\, \mathsf{ff} \wedge [\, x \neq \star\,]\, X)$$

  is satisfied unless that value appears in the trace, leading to a violation of the subformula $[\, x = \star\,]\, \mathsf{ff}$. Recall that a greatest-fixed-point formula is satisfied unless one of its finite unfoldings is violated by the trace under consideration.

As the formulae in the above example indicate, the logic RECHML^d is very expressive. Indeed, it turns out that even some of its fragments that make a restricted use of fixed-point formulae are undecidable.

Theorem 3.1 (Theorems 3–4 in [8])**.** *The satisfiability problem for* CHML^d *and the validity problem for* DISJHML^d *are undecidable.*

The above result, which is shown by adapting and sharpening the reduction used in the proof of [70, Theorem 18], paints a grim decidability picture for RECHML^d. By comparison, the satisfiability and validity problems for RECHML and the modal μ-calculus are EXP-complete [5, 61]. Fortunately, however, as we will now see, those undecidability results do not prevent us from identifying monitorable fragments of that logic.

3.2 What are the monitorable RECHML^d formulae?

Our goal now is to determine which properties over TRC$^d = \mathbb{D}^\omega$ can be monitored and with what guarantees. As in the classic setting we considered in the previous

section, a monitor m can be abstractly characterised by two sets of finite data words $A(m)$ and $R(m)$ included in $\mathbb{D}^*$ that are disjoint and extension closed. The definition of when a monitor accepts or rejects a trace in Trc^d given in Definition 2.1 applies to our current setting mutatis mutandis, as do those of monitor soundness and of the various flavours of completeness with respect to a formula $\varphi \in \text{recHML}^d$ and to a property of traces $T \subseteq \text{Trc}^d$. We say that the above notions are *effective* when m can be computed by a Turing machine from φ or some other finitary description of a property T.

In the finite alphabet case, we saw that all completely monitorable properties of traces can be expressed in the fragment HML, which consists of the recHML formulae without fixed-point operators (Theorem 2.2). The proof of that result from [13] (see Theorem 4.8 in that reference) can be adapted to establish a counterpart of that theorem in the setting of properties of data words. We tame the infinity of the data domain by quotienting finite traces by bijections over $\mathbb{D}$.

Theorem 3.2. *Let $T \subseteq \text{Trc}^d$ be a set of traces that is stable under renamings (that is, for all bijections $\sigma : \mathbb{D} \to \mathbb{D}$, we have that $\sigma(T) = T$). T is completely monitorable* iff *it can be expressed in* HML^d.

Remark. The set $[\![\varphi]\!]$ is stable under renamings for each formula $\varphi \in \text{recHML}^d$. Such sets are called *equivariant* in the theory of nominal sets [32, 71].

Remark. Theorem 3.2 does not hold if we consider the domain $(\mathbb{N}, <)$. Indeed, there, one can define the set $D = \{d_0 d_1 \dots d_n \# w \mid \forall i < j, d_i > d_j\}$, which is completely monitorable, since n is bounded by d_0, but cannot be expressed in HML^d since n depends on d_0.

As Theorem 3.2 indicates, having to detect *all* satisfactions and *all* violations prevents us from monitoring for behaviours that can happen after an unbounded number of steps in a system execution. In what follows, we relax our notion of completeness and focus on satisfaction-completeness, the ability to detect all satisfactions of a property. Results dealing with violation-completeness are obtained by duality.

In Figure 5, we introduce a model of monitors, along with a compositional monitor-synthesis procedure from formulae in cHML^d (defined in Figure 4). We encourage our readers to compare the syntax and semantics of monitors in the setting with data given in Figure 5 with those we introduced in Figure 3. In particular, note that the prefixing operator we employ in the setting with data corresponds to a conditional choice on a Boolean expression b and matches the modalities in recHML^d. Moreover, the construct guess $x.m$ plays the role of quantification over data values in formulae. Since the behaviour of monitors depends on the current values of its data variables, the structural operational semantics of monitors

Syntax

$m, n \in \text{Mon} ::= \mathsf{yes} \mid \mathsf{end} \mid (b).m \mid \mathsf{guess}\ x.m \mid m \oplus n \mid m \otimes n \mid \mathsf{rec}\, x.m \mid x$

Configurations $c \in C ::= (m, \delta) \mid c \odot c$, where $m \in \text{Mon}$ is a monitor, $\delta \in \text{DEnv}$ is a data environment and $\odot$ is either $\oplus$ (parallel OR) or $\otimes$ (parallel AND).

Small-Step Semantics

$$\frac{v \in \{\mathsf{yes}, \mathsf{end}\}}{v,\ \delta \xrightarrow{d} v,\ \delta} \qquad \frac{b\delta[\star \mapsto d] \Downarrow \mathsf{true}}{(b).m,\ \delta \xrightarrow{d} m,\ \delta}\ d \in \mathbb{D} \qquad \frac{b\delta[\star \mapsto d] \Downarrow \mathsf{false}}{(b).m,\ \delta \xrightarrow{d} \mathsf{end},\ \delta}\ d \in \mathbb{D}$$

$$\frac{}{\mathsf{guess}\ x.m,\ \delta \xrightarrow{\tau} m,\ \delta[x \mapsto d]}\ d \in \mathbb{D} \qquad \frac{}{\mathsf{rec}\, x.m,\ \delta \xrightarrow{\tau} m[\mathsf{rec}\, x.m/x],\ \delta}$$

$$\frac{}{m \odot n,\ \delta \xrightarrow{\tau} m,\ \delta \odot n,\ \delta} \qquad \frac{c_1 \xrightarrow{d} c_1' \quad c_2 \xrightarrow{d} c_2'}{c_1 \odot c_2 \xrightarrow{d} c_1' \odot c_2'}$$

$$\frac{c_1 \xrightarrow{\tau} c_1'}{c_1 \odot c_2 \xrightarrow{\tau} c_1' \odot c_2} \qquad \frac{c_2 \xrightarrow{\tau} c_2'}{c_1 \odot c_2 \xrightarrow{\tau} c_1 \odot c_2'}$$

$$\frac{}{\mathsf{yes},\ \delta \otimes c \xrightarrow{\tau} c} \qquad \frac{}{\mathsf{yes},\ \delta \oplus c \xrightarrow{\tau} \mathsf{yes},\ \delta}$$

Monitor Synthesis

$$\mathsf{m}(\mathsf{tt}) = \mathsf{yes} \quad \mathsf{m}(\exists\, \boldsymbol{x}.\varphi) = \mathsf{guess}\ x.\mathsf{m}(\varphi) \quad \mathsf{m}(\langle\, b\,\rangle\, \varphi) = (b).\mathsf{m}(\varphi)$$
$$\mathsf{m}(\varphi \vee \psi) = \mathsf{m}(\varphi) \oplus \mathsf{m}(\psi) \quad \mathsf{m}(\varphi \wedge \psi) = \mathsf{m}(\varphi) \otimes \mathsf{m}(\psi)$$
$$\mathsf{m}(\mathsf{min}\ X.\ (\varphi)) = \mathsf{rec}\, x.\mathsf{m}(\varphi) \qquad \mathsf{m}(X) = x$$

Figure 5: Syntax, small-step semantics and synthesis of monitors.

employs *configurations*, which consist of parallel compositions of monitors, each with its own data environment δ that assigns values to its data variables.

As we did earlier in the classic setting, using the operational semantics of monitors, we can now define the set of finite traces $A(m) \subseteq \mathbb{D}^*$ associated with a monitor m. In what follows, we write $m, \varnothing \stackrel{s}{\Rightarrow} c$ for some monitor m, finite trace $s \in \mathbb{D}^*$ and configuration c when there is a sequence of transitions leading from $m, \varnothing$ to c whose observable content is equal to s.

Definition 3.1. For each $m \in \textsc{Mon}$, we define

$$A(m) = \{s \mid m, \varnothing \stackrel{s}{\Rightarrow} \mathsf{yes}, \delta, \text{ for some } \delta\}.$$

The notions of monitor soundness and satisfaction-completeness are defined using $A(m)$ as we did in the classic setting in Definition 2.2.

We now show that, as the following example indicates in a specific setting, the monitoring system from Figure 5 yields sound and satisfaction-complete monitors for formulae in cHML^d. Note that this fragment of $\textsc{rec}\text{HML}^d$ includes conjunctions, and it can express ff as, for example, the formula $\langle \bot \rangle \mathsf{tt}$ (where $\bot$ stands for $x \neq x$) and (linear-time) necessity modalities.

Example 3.2. Consider a server that issues identifier tokens. Assume that the first token it issues is private and should not be leaked, that is, the server should *not* satisfy the formula φ_{leak} given in (2), which we repeat below for ease of reference:

$$\varphi_{\text{leak}} \stackrel{\text{def}}{=} \exists \boldsymbol{x}.\langle x = \star \rangle \min X.(\langle x = \star \rangle \mathsf{tt} \vee \langle x \neq \star \rangle X).$$

The application of the monitor-synthesis function in Figure 5 to the above formula yields the monitor

$$m_{\text{leak}} \stackrel{\text{def}}{=} \mathsf{m}(\varphi_{\text{leak}}) = \mathsf{guess}\ x.(x = \star).\mathsf{rec}\ x.((x = \star).\mathsf{yes} \oplus (x \neq \star).X).$$

Consider an erroneous execution 110^ω exhibited by the server. The monitor m_{leak} starts in configuration

$$\mathsf{guess}\ x.(x = \star).\mathsf{rec}\ x.((x = \star).\mathsf{yes} \oplus (x \neq \star).X), \varnothing.$$

In its first step, the monitor m_{leak} internally selects a concrete value $d \in \mathbb{D}$ for x. Note that such a value is selected over a possibly infinite domain, reminiscent of the countable nondeterminism studied in the classic paper [23]. Assume that the monitor chooses the value 0 for x. In the next step, the system emits 1 and the monitor checks for the guard $(x = \star)$, which does not hold. Thus, the monitor transitions to the configuration $\mathsf{end}, x \mapsto 0$, where it stays forever. Therefore, that monitor run leads to an inconclusive verdict.

Assume instead that the monitor picks $x = 1$ in its first step. Then, as our readers can check, the execution of the monitor continues and, after observing the prefix 11 of the trace 110^ω, the monitor raises a yes verdict. Thus, the trace is accepted by the monitor, indicating that the system repeats its first action and thus leaks its first data value.

Theorem 3.3 (Theorem 17 in [8])**.** *The monitor $\mathsf{m}(\varphi)$ is sound and satisfaction-complete for each $\varphi \in \mathrm{cHML}^d$.*

Remark. By duality, a similar result holds for formulae in sHML^d, for which one can construct sound and violation-complete monitors.

The monitor model we have used to prove Theorem 3.3 is equivalent to a model of register automata. Register automata were introduced in [59] as *finite-memory automata*. They consist of a finite-state automaton equipped with a finite set of *registers* that can store values from an infinite domain (here, $\mathbb{D}$). A register automaton can compare the value it reads with the content of its registers and move accordingly. We eschew the formal definition of the model that we use to establish the following result and limit ourselves to remarking that it is equivalent to that of [32, Section 1.3], omitting labels, which play no role in our developments. We refer the interested reader to [9, Appendix A.13] for the formal definitions and technical details.

Theorem 3.4. *Let $L \subseteq \mathbb{D}^*$ be an extension-closed language. There exists an alternating register automaton with existential guessing that recognises L if and only if there exists a monitor that accepts exactly the traces in L.*

This result echoes the equivalence between the alternation-free μ-calculus and register tree automata presented in Theorems 3 and 7 in [58].

The correspondence also holds between register automata with no universal (respectively, existential) states and monitors with no occurrences of $\otimes$ (respectively, $\oplus$). Moreover, if one defines $\mathsf{match}(r, b) \stackrel{\text{def}}{=} \mathsf{guess}\ r.(b \wedge r = \star)$, all the above correspondences hold for register automata without guessing and monitors whose $\mathsf{guess}\ r$ construct is replaced with the $\mathsf{match}(r, b)$ one.

Since all those classes of register automata are inequivalent [32, Section 1.5], we know that all those variants of monitors correspond to different classes of properties. Thus, in cHML^d and sHML^d, removing conjunctions or disjunctions reduces expressiveness, and the same holds when replacing existential quantification with a $\mathsf{match}(r, b)$ construct (as defined in [18]). This also shows that deterministic monitors (defined as the counterpart of deterministic register automata) are strictly less expressive than non-deterministic or alternating ones, which invalidates [18, Theorem 18].

A natural question to ask at this point is whether cHML^d suffices to express all the formulae that have sound and satisfaction complete monitors. We now show that, in contrast to the finite alphabet case [13, Proposition 4.18], that fragment is not maximally expressive: there are properties that admit sound and satisfaction-complete monitors that cannot be expressed in cHML^d. Indeed, some formulae containing universal quantifiers *are* monitorable. As an example, consider the property that states that the input is divided into blocks separated by dollar and sharp symbols, and that all data values that appear in the second block appear in the first block, which is formalised as follows:

$$L_{\forall\#\exists\$} = \left\{ d_1 \dots d_k \$ e_1 \dots e_l \# \dots \;\middle|\; \forall 1 \leq j \leq l, \exists 1 \leq i \leq k, d_i = e_j \right\}, \tag{6}$$

That property admits a sound and satisfaction-complete monitor: the monitor reads the first two blocks by waiting to see the \$ and then the #; if this never happens, it means that the input violates the property. Otherwise, the monitor can check that all data values in the second block appear in the first one by processing them one by one, going back and forth.

The property $L_{\forall\#\exists\$}$ cannot be expressed in cHML^d—see [8, Proposition 21]. Intuitively, the length of the second block is unbounded, and its elements have to be compared with elements that appear *before* in the input, so they cannot be manipulated only using existential quantifiers, even with fixed points. Thus, cHML^d is not a maximally expressive fragment whose formulae are monitorable for satisfaction using 'computable monitors'.

However, the property $L_{\forall\#\exists\$}$ *can* be expressed in recHML^d thus:

$$\begin{aligned}
&\varphi_{\forall\#\exists\$} = \exists\, \boldsymbol{x}.\gamma(x) \wedge \forall\, \boldsymbol{x}.\, (\gamma(x) \vee \psi(x))\,, \text{ where:}\\
&\gamma(x) = \mathsf{min}\, X.\, (\langle\, \star \neq \$\,\rangle\, X \vee \langle\, \star = \$\,\rangle\, \mathsf{min}\, Y.\, (\langle\, \star = \#\,\rangle\, \mathsf{tt} \vee \langle\, \star \neq x\,\rangle\, Y))\\
&\psi(x) = \mathsf{min}\, Z.\, (\langle\, \star = x\,\rangle\, \mathsf{tt} \vee \langle\, \star \neq \$\,\rangle\, Z).
\end{aligned}$$

The formula $\gamma(x)$ is called the *guard*, and sets a monitorable bound on the maximal position where a candidate x violating the formula ψ can be found. This way, once the bound is found, the monitor knows that the subsequent data values that appear need not be checked. Here, it expresses that the trace starts with two blocks—ended by \$ and #, respectively—and that x does not appear in the second block: first, look for a \$; once it is found, look for a #, and if in the meantime x is encountered, the formula cannot recurse and therefore rejects.

The formula $\psi(x)$ is the universally quantified property, and since we are looking for satisfactions, its universal quantification has to be limited to finitely many values to ensure that it has a finite witness, hence the disjunction with the guard. Here, it expresses that x appears in the first block. Summing up, the conjunct $\exists\, \boldsymbol{x}.\gamma(x)$ ensures that a trace has the form $w_1\$w_2\#u$ for some $w_1, w_2 \in \mathbb{D}^*$

and $u \in \textsc{Trc}^d$, while the conjunct $\forall\, \boldsymbol{x}.(\gamma(x) \vee \psi(x))$ yields that every $d \in \mathbb{D}$ occurring in w_2 also occurs in w_1.

Based on a substantial generalisation of the pattern highlighted by the guarded formula $\varphi_{\forall\#\exists\$}$, in [8, Theorem 26] we offer a characterisation of the collection of formulae in $\textsc{recHML}^d$ without greatest fixed points that can be monitored in a sound and satisfaction-complete fashion by computable monitors. The definition of that fragment and the proofs of the technical results for it are too intricate to be discussed in this survey article; they can be found in [8, Section 4.1] and [9]. It turns out that the fragment we identify does not express all the formulae in $\textsc{recHML}^d$ that have sound and satisfaction-complete monitors. However, in a precise sense, this is the best we can do: there is no maximally monitorable fragment of $\textsc{recHML}^d$ that is effective (see [8, Corollary 29]).

4 Exhibit C: Centralised and decentralised monitoring of hyperproperties

The last leg of our journey brings us to the very active study of runtime monitoring for hyperproperties—see, for instance, [2, 21, 33, 34, 40, 41, 48, 53]. *Hyperproperties* were introduced in [44] as sets of *hypertraces*, which are sets of traces that may be seen as describing different system executions or the contributions of different sequential processes to a system execution. Since their introduction, hyperproperties have become a fundamental, trace-based formalism for expressing security and privacy properties, verified using static and dynamic techniques [21,31,33,34,40,48] implemented in a variety of tools [30,47]. Several extensions of temporal logics have been proposed in the literature to describe hyperproperties. Examples of such logics include HyperLTL, HyperCTL* [43], Hyper^{2}LTL [31], and those based on variations on the μ-calculus [2, 53].

In our work [3, 4], we have used a view of hypertraces that differs from the classic, set-based one from [44]. As in Section 2, we assume a finite set of actions $\textsc{Act}$ that contains at least two actions and write $\textsc{Trc}$ for the collection of infinite traces over $\textsc{Act}$. Given a finite and non-empty set $\mathcal{L}$ of *locations*, ranged over by ℓ, a *hypertrace* T over $\mathcal{L}$ is a function from $\mathcal{L}$ to $\textsc{Trc}$. The set of hypertraces on $\mathcal{L}$ is denoted by $\textsc{HTrc}_{\mathcal{L}}$.

Intuitively, a hypertrace describes an execution of a (distributed) system with $|\mathcal{L}|$ users; every user is located at a unique location chosen from $\mathcal{L}$ and each user is mapped to the trace they perform.

For $t, t' \in \textsc{Trc}$, we write $t \xrightarrow{a} t'$ whenever $t = at'$. We call a function $A : \mathcal{L} \to \textsc{Act}$ a hyperaction. For $T, T' \in \textsc{HTrc}_{\mathcal{L}}$, we write $T \xrightarrow{A} T'$ whenever $T(\ell) \xrightarrow{A(\ell)} T'(\ell)$,

for every $\ell \in \mathcal{L}$. Notice that, for each T, there is a *unique* pair A and T' such that $T \xrightarrow{A} T'$: more precisely, for every $\ell \in \mathcal{L}$, we have that $A(\ell) = a$ and $T'(\ell) = t'$, whenever $T(\ell) = at'$. We denote the A and T' just defined by $\mathrm{hd}(T)$ and $\mathrm{tl}(T)$, respectively.

The notation we just presented reflects the classic synchronous view of hypertraces and hyperproperties, according to which, at every step, a system proceeds by performing an action in Act at each of its locations. In this setting, a *finite* hypertrace S is a function from $\mathcal{L}$ to Act^n, for some $n \geq 0$. Each finite hypertrace S results from the pointwise concatenation of some hyperactions $A_1, \ldots, A_n : \mathcal{L} \to \mathrm{Act}$, for some $n \geq 0$, thus:

$$S(\ell) = A_1 \cdots A_n(\ell) \quad = \quad A_1(\ell) \cdots A_n(\ell).$$

The concatenation of a finite hypertrace and a (finite) hypertrace is defined analogously in pointwise fashion. We say that a finite hypertrace S is a prefix of a hypertrace T if $S(\ell)$ is a prefix of $T(\ell)$ for each $\ell \in \mathcal{L}$—that is, there is some hypertrace U such that $T(\ell) = S(\ell)U(\ell)$, for each $\ell \in \mathcal{L}$. A set of finite hypertraces is extension closed if whenever it contains some finite hypertrace S, then it also contains SW for each finite hypertrace W.

4.1 Syntax and semantics of Hyper-recHML

We adopt a variation on recHML, which we call Hyper-recHML, as the logic to specify *hyperproperties*. We assume two disjoint and countably infinite sets Π and V of *location variables* and *logical variables*, ranged over by π and X, respectively. Formulae of Hyper-recHML are constructed as follows:

$$\begin{aligned} \varphi \quad ::= \quad & \mathsf{tt} \mid \mathsf{ff} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \min X.\varphi \mid \max X.\varphi \mid X \mid \\ & \exists \pi.\varphi \mid \forall \pi.\varphi \mid \pi = \pi \mid \pi \neq \pi \mid [a_\pi]\varphi \mid \langle a_\pi \rangle \varphi. \end{aligned}$$

The new constructs with respect to those used in recHML are existential and universal quantification over locations, equality and inequality tests on location variables, and location-variable-indexed versions of the usual Hennessy-Milner modalities where $[a_\pi]$ stands for 'necessarily after a at the location bound to π', and $\langle a_\pi \rangle$ denotes 'possibly after a at the location bound to π'. Using these constructs, one can express properties such as 'the trace observed at each location starts with an a' (as $\forall \pi.\langle a_\pi \rangle \mathsf{tt}$) and 'there are two different locations whose trace starts with an a' (as $\exists \pi_1.\exists \pi_2.\langle a_{\pi_1} \rangle \mathsf{tt} \wedge \langle a_{\pi_2} \rangle \mathsf{tt} \wedge \pi_1 \neq \pi_2$).

The usual notions of closed and guarded formulae apply as in the previous sections and we tacitly restrict ourselves to those formulae. Moreover, we consider

$$[\![\mathsf{tt}]\!]^{\rho}_{\sigma} = \mathrm{HTRC}_{\mathcal{L}} \qquad [\![\mathsf{ff}]\!]^{\rho}_{\sigma} = \emptyset \qquad [\![X]\!]^{\rho}_{\sigma} = \rho(X)$$

$$[\![\varphi \wedge \varphi']\!]^{\rho}_{\sigma} = [\![\varphi]\!]^{\rho}_{\sigma} \cap [\![\varphi']\!]^{\rho}_{\sigma}$$

$$[\![\varphi \vee \varphi']\!]^{\rho}_{\sigma} = [\![\varphi]\!]^{\rho}_{\sigma} \cup [\![\varphi']\!]^{\rho}_{\sigma}$$

$$[\![\mathsf{max}\, X.\psi]\!]^{\rho}_{\sigma} = \bigcup \{\mathcal{S} \mid \mathcal{S} \subseteq \mathrm{HTRC}_{\mathcal{L}} \text{ and } \mathcal{S} \subseteq [\![\psi]\!]^{\rho[x \mapsto \mathcal{S}]}_{\sigma}\}$$

$$[\![\mathsf{min}\, X.\psi]\!]^{\rho}_{\sigma} = \bigcap \{\mathcal{S} \mid \mathcal{S} \subseteq \mathrm{HTRC}_{\mathcal{L}} \text{ and } \mathcal{S} \supseteq [\![\psi]\!]^{\rho[x \mapsto \mathcal{S}]}_{\sigma}\}$$

$$[\![\exists \pi.\varphi]\!]^{\rho}_{\sigma} = \bigcup_{\ell \in \mathcal{L}} [\![\varphi]\!]^{\rho}_{\sigma[\pi \mapsto \ell]}$$

$$[\![\forall \pi.\varphi]\!]^{\rho}_{\sigma} = \bigcap_{\ell \in \mathcal{L}} [\![\varphi]\!]^{\rho}_{\sigma[\pi \mapsto \ell]}$$

$$[\![\pi = \pi']\!]^{\rho}_{\sigma} = \begin{cases} \mathrm{HTRC}_{\mathcal{L}} & \text{if } \sigma(\pi) = \sigma(\pi') \\ \emptyset & \text{otherwise} \end{cases}$$

$$[\![\pi \neq \pi']\!]^{\rho}_{\sigma} = \begin{cases} \mathrm{HTRC}_{\mathcal{L}} & \text{if } \sigma(\pi) \neq \sigma(\pi') \\ \emptyset & \text{otherwise} \end{cases}$$

$$[\![[a_\pi]\varphi]\!]^{\rho}_{\sigma} = \{T \mid \mathrm{hd}(T)(\sigma(\pi)) = a \text{ implies } \mathrm{tl}(T) \in [\![\varphi]\!]^{\rho}_{\sigma}\}$$

$$[\![\langle a_\pi \rangle \varphi]\!]^{\rho}_{\sigma} = \{T \mid \mathrm{hd}(T)(\sigma(\pi)) = a \ \wedge \ \mathrm{tl}(T) \in [\![\varphi]\!]^{\rho}_{\sigma})\}$$

Figure 6: The semantics of Hyper-recHML.

formulae whose bound location variables are all pairwise distinct (and different from the free variables, when we need to define notions over open fomulae).

The semantics of formulae φ in Hyper-recHML is defined over $\mathrm{HT}_{\mathrm{RC}_{\mathcal{L}}}$ through the function $[\![-]\!]^{\rho}_{\sigma}$, as shown in Figure 6, by exploiting two partial functions: $\rho\colon V \rightharpoonup 2^{\mathrm{HT}_{\mathrm{RC}_{\mathcal{L}}}}$, which assigns a set of hypertraces over $\mathcal{L}$ to all free logical variables occurring in φ, and $\sigma\colon \Pi \rightharpoonup \mathcal{L}$, which assigns a location to all free location variables in φ. In what follows, we tacitly assume that the free logical and location variables in a formula φ are always included in the domains of ρ and σ, respectively. The function ρ gives the semantics of logical variables and σ keeps track of the location associated with each location variable. The function σ is extended every time a new variable π is introduced (through $\exists\pi$ or $\forall\pi$) to check whether the body of the quantification holds for some or for all locations.

All the constructs have the expected semantics. In particular, a formula $\langle a_\pi\rangle\varphi$ holds true at hypertrace T if the trace in T at the location bound to π starts with an a and $\mathrm{tl}(T)$ satisfies φ; by contrast, a formula $[a_\pi]\varphi$ can also hold true if the trace in T at the location associated to π does not start with an a.

Whenever φ is *closed* (i.e., without any free variable), the semantics is given by $[\![\varphi]\!]^{\emptyset}_{\emptyset}$, where $\emptyset$ denotes the partial function with empty domain. Notationally, we shall simply write $[\![\varphi]\!]$ instead of $[\![\varphi]\!]^{\emptyset}_{\emptyset}$. We say that T satisfies the closed formula φ if $T \in [\![\varphi]\!]$.

Example 4.1. For example, consider the set of actions $\{a, b\}$; then, the hyperproperty

$$\varphi_a = \forall\pi.\max X.(\langle b_\pi\rangle X \ \vee\ \exists\pi'.(\pi' \neq \pi \ \wedge\ \langle a_{\pi'}\rangle X)) \tag{7}$$

is a consensus-type property stating that, at every position of every trace, whenever there is an a there is another trace that also has a. Using the definition of the semantics of the logic, it is not hard to see that the hypertrace T_1 over the set of locations $\{\ell_1, \ell_2, \ell_3\}$ that maps ℓ_1 to a^ω, ℓ_2 to ba^ω and ℓ_3 to $(ba)^\omega$ does not satisfy the property φ_a: what breaks the property is the first position. On the other hand, the hypertrace T_2 that maps ℓ_1 to a^ω, ℓ_2 to $(ab)^\omega$ and ℓ_3 to $(ba)^\omega$ does satisfy φ_a because at each position there are two traces that exhibit an a. The hypertrace T_3 that maps each location to b^ω also satisfies φ_a since no a is ever observed.

As argued in [3, Section 2.2], Hyper-recHML is more expressive than HyperLTL and HyperCTL*. This additional expressiveness is also present in the fragments of that logic for which we can synthesise sound and violation-complete monitors.

4.2 What are the monitorable Hyper-recHML formulae?

Following our presentation in the classic setting (Section 2) and in the one with data (Section 3), we can abstractly characterise an irrevocable monitor m for some

$$v \xrightarrow{A} v \qquad \frac{A(\ell) = a}{a_\ell.m \xrightarrow{A} m} \qquad \frac{A(\ell) \neq a}{a_\ell.m \xrightarrow{A} \mathsf{end}} \qquad \frac{m[\mathsf{rec}\, x.m/x] \xrightarrow{A} m'}{\mathsf{rec}\, x.m \xrightarrow{A} m'} \qquad \frac{m \xrightarrow{A} m'}{m + n \xrightarrow{A} m'}$$

$$\frac{n \xrightarrow{A} n'}{m + n \xrightarrow{A} n'} \qquad \frac{m \xrightarrow{A} m' \qquad n \xrightarrow{A} n'}{m \odot n \xrightarrow{A} m' \odot n'}$$

Figure 7: The operational semantics for centralised monitors, where $\odot \in \{\otimes, \oplus\}$.

hyperproperty by means of two disjoint sets of finite hypertraces $A(m)$ and $R(m)$ that are extension closed. Using those sets, we can define when a monitor m is sound and violation- or satisfaction-complete for some property $\varphi \in$ Hyper-recHML by mimicking Definition 2.2. In what follows, we focus solely on monitors that detect violations and therefore we identify a monitor with the set $R(m)$ of finite hypertraces that it rejects.

Example 4.2. Assume that $\mathcal{L} = \{\ell_1, \ell_2\}$ and that $\text{Act} = \{a, b\}$. Let m be a monitor such that $R(m)$ consists of all the finite hypertraces S such that $S(\ell_1)$ or $S(\ell_2)$ starts with a b. Then m is sound and violation-complete for the formula $\forall\pi.\langle a_\pi\rangle\mathsf{tt}$.

Following the lead of the work we discussed in the previous sections, we now present a monitoring system that is based on centralised, omniscient monitors that can monitor for a collection of hyperproperties expressed in a fragment of Hyper-recHML.

The set of *centralised monitors* CMon, ranged over by m, n, is given by the following grammar:

$$m ::= \mathsf{yes} \mid \mathsf{no} \mid \mathsf{end} \mid a_\ell.m \mid m + m \mid m \oplus m \mid m \otimes m \mid \mathsf{rec}\, x.m \mid x.$$

Our readers are already familiar with nearly all the constructs for monitors in CMon. The only new ingredient is the prefixed monitor $a_\ell.m$, which checks if location ℓ is currently performing an a and, in that case, continues as m.

The operational semantics of centralised monitors is given in Figure 7 and follows the intuitive explanations we gave earlier for the different operators. In particular, a monitor that waits for an action at some location (as prescribed by a_ℓ) can either see that action at ℓ (as stated by an hyperaction A) and proceed in its observation, or it does not observe that action and stops its monitoring activity, by reporting end. Recursive monitors should be unfolded to evolve. A

$$v \Rrightarrow v \qquad \frac{m \Rrightarrow \mathsf{end} \quad n \Rrightarrow \mathsf{end}}{m \odot n \Rrightarrow \mathsf{end}} \qquad \frac{m \Rrightarrow \mathsf{yes}}{m \oplus n \Rrightarrow \mathsf{yes}} \qquad \frac{m \Rrightarrow \mathsf{no}}{m \otimes n \Rrightarrow \mathsf{no}}$$

$$\frac{m \Rrightarrow v}{m + n \Rrightarrow v} \qquad \frac{m \Rrightarrow \mathsf{no} \quad n \Rrightarrow v}{m \oplus n \Rrightarrow v} \qquad \frac{m \Rrightarrow \mathsf{yes} \quad n \Rrightarrow v}{m \otimes n \Rrightarrow v} \qquad \frac{m[\mathsf{rec}\, x.m/x] \Rrightarrow v}{\mathsf{rec}\, x.m \Rrightarrow v}$$

Figure 8: Verdict evaluation for centralised monitors (up to commutativity of $+$, $\otimes$, and $\oplus$).

non-deterministic choice $m + n$ can initially behave like m and discard n in doing so or vice versa. Finally, once issued, a verdict never changes; possibly different verdicts obtained in different parallel branches of the monitor can be composed conjunctively or disjunctively. This is represented by the judgement $\Rrightarrow$ defined by the rules in Figure 8. Amongst other things, those rules state that yes and no are the absorbing elements for $\oplus$ and $\otimes$, respectively, and the neutral elements for $\otimes$ and $\oplus$, respectively, that verdict evaluation of a non-deterministic monitor is non-deterministic as well, and that recursive monitors must be unfolded before they can be properly evaluated.

As our attentive readers might have noticed already, the aforementioned syntax of centralised monitors is very general and allows one to write monitors that can omit inconsistent verdicts. For example, the monitor $\mathsf{yes} + \mathsf{no}$ can immediately report both a yes and a no verdict via the transitions $\mathsf{yes} + \mathsf{no} \Rrightarrow \mathsf{yes}$ and $\mathsf{yes} + \mathsf{no} \Rrightarrow \mathsf{no}$, respectively. In what follows, we tacitly restrict ourselves to consistent monitors. Our monitor-synthesis procedures only generate consistent monitors from formulae in Hyper-recHML.

The transition relation $\rightarrow$ can be extended to finite hypertraces by stipulating that $m \xrightarrow{A_1 \cdots A_n} m'$ holds if and only if the pair of monitors (m, m') is contained in the composition of the relations $\xrightarrow{A_1}, \ldots, \xrightarrow{A_n}$. Using the operational semantics of monitors given in Figures 7-8, we can now associate sets of finite hypertraces $A(m)$ and $R(m)$ to each centralised monitor m thus:

$$\begin{aligned} A(m) &= \{A_1 \cdots A_n \mid m \xrightarrow{A_1 \cdots A_n} m' \Rrightarrow \mathsf{yes}, \text{ for some } m'\} \text{ and} \\ R(m) &= \{A_1 \cdots A_n \mid m \xrightarrow{A_1 \cdots A_n} m' \Rrightarrow \mathsf{no}, \text{ for some } m'\}. \end{aligned}$$

Example 4.3. Let us revisit the setting of Example 4.2, where we assumed that

$$
\begin{aligned}
\mathsf{cm}(\mathsf{tt})_\sigma &= \mathsf{yes} \\
\mathsf{cm}(\mathsf{ff})_\sigma &= \mathsf{no} \\
\mathsf{cm}(X)_\sigma &= x \\
\mathsf{cm}(\max X.\varphi)_\sigma &= \mathsf{rec}\, x.\mathsf{cm}(\varphi)_\sigma \\
\mathsf{cm}(\varphi \wedge \varphi')_\sigma &= \mathsf{cm}(\varphi)_\sigma \otimes \mathsf{cm}(\varphi')_\sigma \\
\mathsf{cm}(\varphi \vee \varphi')_\sigma &= \mathsf{cm}(\varphi)_\sigma \oplus \mathsf{cm}(\varphi')_\sigma \\
\mathsf{cm}(\forall \pi.\varphi)_\sigma &= \bigotimes_{\ell \in \mathcal{L}} \mathsf{cm}(\varphi)_{\sigma[\pi \mapsto \ell]} \\
\mathsf{cm}(\exists \pi.\varphi)_\sigma &= \bigoplus_{\ell \in \mathcal{L}} \mathsf{cm}(\varphi)_{\sigma[\pi \mapsto \ell]} \\
\mathsf{cm}(\pi = \pi')_\sigma &= \begin{cases} \mathsf{yes} \text{ if } \sigma(\pi) = \sigma(\pi') \\ \mathsf{no} \text{ otherwise} \end{cases} \\
\mathsf{cm}(\pi \neq \pi')_\sigma &= \begin{cases} \mathsf{yes} \text{ if } \sigma(\pi) \neq \sigma(\pi') \\ \mathsf{no} \text{ otherwise} \end{cases} \\
\mathsf{cm}([a_\pi]\varphi)_\sigma &= a_{\sigma(\pi)}.\mathsf{cm}(\varphi)_\sigma + \sum_{b \neq a} b_{\sigma(\pi)}.\mathsf{yes} \\
\mathsf{cm}(\langle a_\pi \rangle \varphi)_\sigma &= a_{\sigma(\pi)}.\mathsf{cm}(\varphi)_\sigma + \sum_{b \neq a} b_{\sigma(\pi)}.\mathsf{no}
\end{aligned}
$$

Figure 9: Centralised monitor synthesis.

$\mathcal{L} = \{\ell_1, \ell_2\}$ and that $\text{Act} = \{a, b\}$. Consider the monitor

$$m = (a_{\ell_1}.\mathsf{yes} + b_{\ell_1}.\mathsf{no}) \otimes (a_{\ell_2}.\mathsf{yes} + b_{\ell_2}.\mathsf{no}).$$

It is not hard to see that $R(m)$ consists of all the finite hypertraces S such that $S(\ell_1)$ or $S(\ell_2)$ starts with a b. Therefore, as we saw in Example 4.2, m is sound and violation-complete for the formula $\forall\pi.\langle a_\pi\rangle\mathsf{tt}$.

We now show how to automatically construct sound and violation-complete monitors from formulae in Hyper-recHML without least fixed-point operators. The definition of the synthetised monitor is given by induction on a formula φ and is parametrised by a partial function σ, assigning a location to all the free location variables of φ; when φ is closed, we consider $\mathsf{cm}(\varphi)_\emptyset$. The formal definition is given in Figure 9. A monitor synthetized from a greatest-fixed-point formula is itself recursive and intuitively checks whether some unfolding of the formula is violated. The monitor for $\varphi \wedge \varphi'$ (respectively, $\varphi \vee \varphi'$) is obtained as the 'parallel and' (respectively, the 'parallel or') of the monitors synthesised from φ and φ'. Universal and existential quantifiers are treated as conjunctions and disjunctions, respectively, and use the function σ to assign values to the newly introduced location variable. Finally, the monitors for formulae of the form $[a_\pi]\varphi$ and $\langle a_\pi\rangle\varphi$ look for an occurrence of action a at the location bound to π in σ; if such action is observed at that location, then the monitor proceeds by checking the rest of the formula, otherwise the monitor for $\langle a_\pi\rangle\varphi$ returns no whereas the monitor for $[a_\pi]\varphi$ returns yes. Notice that we are using the choice operator '+' here to consider only one of the possible observations (corresponding to the action occurring at the location $\sigma(\pi)$) and discarding all the other ones.

The following result states the correctness of the monitor-synthesis function given in Figure 9—see Theorems 3.5 and 3.8 in [4] for its proof.

Theorem 4.1. *Let φ be a closed formula in Hyper-recHML that does not contain occurrences of least-fixed-point operators and let $T \in \text{HTrc}_{\mathcal{L}}$. Then $\mathsf{cm}(\varphi)_\emptyset$ rejects T iff $T \notin [\![\, \varphi \,]\!]$.*

The above results tells us how to generate sound and violation-complete centralised monitors for a fragment of our touchstone logic Hyper-recHML. However, when verifying a distributed system, having a central authority that performs any type of runtime verification is a strong assumption, as it reduces the appeal of distribution, creates single points of failure during verification and can pose problems in storing all the traces locally, especially in light of the wide availability of multi-core systems. Thus, in [3, 4, Section 4], we studied to what extent hyperproperties can be monitored by *decentralised* monitors; these avoid high contentions (leading to vastly improved scalability [17]) and also offer better privacy guarantees (whenever they are stationed locally at the nodes where the respective traces are generated).

Intuitively, in a decentralised-monitoring setting, we associate monitors defined as in Figure 3, with locations. A monitor m located at ℓ, denoted by $[m]_\ell$, observes only actions required to happen at ℓ, thus allowing the processing of events to occur locally. As in [2], located monitors of the form $[m]_\ell$ are nodes in a structure that resembles a Boolean circuit, which allows them to combine their individual verdicts into a global one. However, unlike in the aforementioned reference, in [3,4] we introduce the possibility for monitors to communicate with one another and coordinate their behaviour when checking for properties involving several quantifiers, and therefore the interplay between various traces in a hypertrace. To this end, we extend the syntax of local monitors from Figure 3 with a new prefixing operator of the form $c.m$, where c is a *communication action*. The form of monitor communication we studied in [3, 4] was *multicast* and involved communication actions of the form $(!G, \gamma)$, for sending γ to all locations in the group G, and $(?G, \gamma)$, for receiving γ from any monitor from the set of locations G.

The details of the operational semantics of decentralised monitors are rather involved, and so is the procedure for synthesising decentralised monitors from formulae in Hyper-recHML without least fixed points that are Boolean combinations of formulae in prenex form. (See [3, Section 4.2] for the precise definition of that fragment of Hyper-recHML.) We therefore refer our readers to the aforementioned reference for details. Here we limit ourselves to mentioning that the proof of correctness in the decentralised case is considerably more technical than the corresponding proof of Theorem 4.1, due to the intricate communication semantics of decentralised monitors. To address the resulting technical challenges, we again rely on classic tools from concurrency theory to develop a proof strategy where we prove the correctness of the decentralised monitor synthesis procedure using the centralised one as a yardstick. More precisely, in [3, Definition 13] we identify six properties of a decentralised monitor synthesis that make it 'principled' and we show that, when a decentralised monitor synthesis is principled, the centralised and decentralised monitors synthesised from a formula are related by a suitable notion of weak bisimulation (see [3, Theorem 14]). Apart from supporting the definition of decentralised monitor synthesis procedures, this result allows us to reduce the correctness of our decentralised monitor synthesis to that of the centralised one, which can in turn drive the definition of further synthesis procedures in future work.

5 Concluding remarks

The goal of this article was to provide an introduction to some of our work on runtime monitoring, highlighting the role that classic concurrency theory has

played in our technical developments. It is not up to us to assess the value of what we have achieved thus far for the runtime monitoring community, but we hope that some of the readers of this article will be enticed to study the technical contributions in the papers mentioned in the bibliography in more detail.

Our research journey in the theoretical foundations of runtime monitoring is by no means over and much remains to be done. For example, our work presented in Sections 3–4 falls short of providing theoretical developments that are as general and complete as those in the classic setting studied in Section 2. For example, we still need to develop an elegant (process-algebraic) model of monitors that are sound and satisfaction-complete for the maximally-expressive, guarded fragment of RECHML^d identified in [8]. Moreover, in the setting of Hyper-recHML, we are still missing maximality and expressiveness results. For instance, it is natural to wonder whether the fragment of that logic for which we synthesised sound and violation-complete monitors can express all the hyperproperties for which such monitors exist. Moreover, we do not know yet whether the decentralised monitors with multicast communication we studied in [3, 4] are less powerful than the centralised monitors. And does the expressive power of decentralised monitors depend on the form of communication they employ? We are currently working on these questions and on some others through the lens of concurrency theory, and hope to report on any answers we might obtain in future papers. To quote the title of an ERC project led by Thomas A. Henzinger on the topic of runtime monitoring, 'VAMOS!'[3]

Acknowledgements The work reported in this article has been partially supported by grants No. 163406-051, 184940-051 and 217987 of the Icelandic Research Fund. We are most grateful to that funding agency and the Reykjavik University Research Fund for supporting theoretical research in concurrency theory and runtime monitoring over many years.

We thank Marino Miculan and Nobuko Yoshida for their interest in our work and for commissioning this paper.

References

[1] Yael Abarbanel, Ilan Beer, Leonid Gluhovsky, Sharon Keidar, and Yaron Wolfsthal. FoCs: Automatic generation of simulation checkers from formal specifications. In E. Allen Emerson and A. Prasad Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 538–542. Springer, 2000.

[3] See `https://ista.ac.at/en/vigilant-algorithmic-monitoring-of-software/`.

[2] Luca Aceto, Antonis Achilleos, Elli Anastasiadi, and Adrian Francalanza. Monitoring hyperproperties with circuits. In Mohammad Reza Mousavi and Anna Philippou, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 42nd IFIP WG 6.1 International Conference, FORTE 2022*, volume 13273 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2022.

[3] Luca Aceto, Antonis Achilleos, Elli Anastasiadi, Adrian Francalanza, Daniele Gorla, and Jana Wagemaker. Centralized vs decentralized monitors for hyperproperties. In Rupak Majumdar and Alexandra Silva, editors, *35th International Conference on Concurrency Theory, CONCUR 2024*, volume 311 of *LIPIcs*, pages 4:1–4:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[4] Luca Aceto, Antonis Achilleos, Elli Anastasiadi, Adrian Francalanza, Daniele Gorla, and Jana Wagemaker. Centralized vs decentralized monitors for hyperproperties. *ACM Trans. Comput. Log.*, 27(1):2:1–2:57, 2026.

[5] Luca Aceto, Antonis Achilleos, Elli Anastasiadi, Adrian Francalanza, and Anna Ingólfsdóttir. Complexity results for modal logic with recursion via translations and tableaux. *Log. Methods Comput. Sci.*, 20(3), 2024.

[6] Luca Aceto, Antonis Achilleos, Elli Anastasiadi, and Anna Ingólfsdóttir. Axiomatizing recursion-free, regular monitors. *J. Log. Algebraic Methods Program.*, 127:100778, 2022.

[7] Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Léo Exibard, Adrian Francalanza, and Anna Ingólfsdóttir. A monitoring tool for linear-time μHML. *Sci. Comput. Program.*, 232:103031, 2024.

[8] Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Léo Exibard, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Monitorability for the modal mu-calculus over systems with data: From practice to theory. In Patricia Bouyer and Jaco van de Pol, editors, *36th International Conference on Concurrency Theory, CONCUR 2025*, volume 348 of *LIPIcs*, pages 4:1–4:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. Full version available at `https://doi.org/10.48550/arXiv.2506.06172`.

[9] Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Léo Exibard, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Monitorability for the modal mu-calculus over systems with data: From practice to theory. *CoRR*, abs/2506.06172, 2025.

[10] Luca Aceto, Antonis Achilleos, Adrian Francalanza, and Anna Ingólfsdóttir. Monitoring for silent actions. In Satya V. Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017*, volume 93 of *LIPIcs*, pages 7:1–7:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[11] Luca Aceto, Antonis Achilleos, Adrian Francalanza, and Anna Ingólfsdóttir. A framework for parameterized monitorability. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures - 21st International*

Conference, FOSSACS 2018, volume 10803 of *Lecture Notes in Computer Science*, pages 203–220. Springer, 2018.

[12] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Sævar Örn Kjartansson. Determinizing monitors for HML with recursion. *J. Log. Algebraic Methods Program.*, 111:100515, 2020.

[13] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karolina Lehtinen. Adventures in monitorability: From branching to linear time and back again. *Proc. ACM Program. Lang.*, 3(POPL):52:1–52:29, 2019. Full version available at `https://arxiv.org/abs/1902.00435`.

[14] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karolina Lehtinen. The cost of monitoring alone. In Ezio Bartocci, Rance Cleaveland, Radu Grosu, and Oleg Sokolsky, editors, *From Reactive Systems to Cyber-Physical Systems - Essays Dedicated to Scott A. Smolka on the Occasion of His 65th Birthday*, volume 11500 of *Lecture Notes in Computer Science*, pages 259–275. Springer, 2019.

[15] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karolina Lehtinen. The best a monitor can do. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic, CSL 2021*, volume 183 of *LIPIcs*, pages 7:1–7:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[16] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karolina Lehtinen. An operational guide to monitorability with applications to regular properties. *Softw. Syst. Model.*, 20(2):335–361, 2021.

[17] Luca Aceto, Duncan Paul Attard, Adrian Francalanza, and Anna Ingólfsdóttir. Runtime instrumentation for reactive components. In Jonathan Aldrich and Guido Salvaneschi, editors, *38th European Conference on Object-Oriented Programming, ECOOP 2024*, volume 313 of *LIPIcs*, pages 2:1–2:33. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[18] Luca Aceto, Ian Cassar, Adrian Francalanza, and Anna Ingólfsdóttir. On runtime enforcement via suppressions. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018*, volume 118 of *LIPIcs*, pages 34:1–34:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[19] Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge Univ. Press, 2007.

[20] Antonis Achilleos, Adrian Francalanza, and Jasmine Xuereb. If at first you don't succeed: Extended monitorability through multiple executions. In *40th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2025*, pages 636–650. IEEE, 2025.

[21] Shreya Agrawal and Borzoo Bonakdarpour. Runtime verification of k-safety hyperproperties in HyperLTL. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016*, pages 239–252. IEEE Computer Society, 2016.

[22] Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, Reiner Hähnle, and Mattias Ulbrich, editors. *Deductive Software Verification: Future Perspectives - Reflections on the Occasion of 20 Years of KeY*, volume 12345 of *Lecture Notes in Computer Science*. Springer, 2020.

[23] Krzysztof R. Apt and Gordon D. Plotkin. Countable nondeterminism and random assignment. *J. ACM*, 33(4):724–767, 1986.

[24] Duncan Paul Attard, Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Better late than never or: Verifying asynchronous components at runtime. In Kirstin Peters and Tim A. C. Willemse, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 41st IFIP WG 6.1 International Conference, FORTE 2021*, volume 12719 of *Lecture Notes in Computer Science*, pages 207–225. Springer, 2021.

[25] Duncan Paul Attard, Ian Cassar, Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. A runtime monitoring tool for actor-based systems. In Simon Gay and Antonio Ravara, editors, *BETTY Book*, pages 49–76. River Publishers, 2017. Chapter 3. Available at `http://www.riverpublishers.com/downloadchapter.php?file=RP_9788793519817C3.pdf`.

[26] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[27] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Rule-based runtime verification. In Bernhard Steffen and Giorgio Levi, editors, *Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004*, volume 2937 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2004.

[28] Ezio Bartocci and Yliès Falcone, editors. *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*. Springer, 2018.

[29] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[30] Raven Beutner and Bernd Finkbeiner. Software verification of hyperproperties beyond k-safety. In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022*, volume 13371 of *Lecture Notes in Computer Science*, pages 341–362. Springer, 2022.

[31] Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Niklas Metzger. Monitoring second-order hyperproperties. In Mehdi Dastani, Jaime Simão Sichman, Natasha Alechina, and Virginia Dignum, editors, *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024*, pages 180–188. International Foundation for Autonomous Agents and Multiagent Systems / ACM, 2024.

[32] Mikołaj Bojańczyk. *Slightly Infinite Sets*. Online book draft, 2019. Available at `https://www.mimuw.edu.pl/~bojan/paper/atom-book`.

[33] Borzoo Bonakdarpour and Bernd Finkbeiner. Runtime verification for HyperLTL. In Yliès Falcone and César Sánchez, editors, *Runtime Verification - 16th International Conference, RV 2016*, volume 10012 of *Lecture Notes in Computer Science*, pages 41–45. Springer, 2016.

[34] Borzoo Bonakdarpour, César Sánchez, and Gerardo Schneider. Monitoring hyperproperties by combining static analysis and runtime verification. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Verification - 8th International Symposium, ISoLA 2018*, volume 11245 of *Lecture Notes in Computer Science*, pages 8–27. Springer, 2018.

[35] Florian Bruse, Oliver Friedmann, and Martin Lange. On guarded transformation in the modal μ-calculus. *Log. J. IGPL*, 23(2):194–216, 2015.

[36] Filip Cano, Thomas A. Henzinger, and Konstantin Kueffner. Algorithmic fairness: A runtime perspective. In Bettina Könighofer and Hazem Torfah, editors, *Runtime Verification - 25th International Conference, RV 2025*, volume 16087 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2025.

[37] Ian Cassar, Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. eAOP: an aspect oriented programming framework for Erlang. In Natalia Chechina and Scott Lystig Fritchie, editors, *Proceedings of the 16th ACM SIGPLAN International Workshop on Erlang*, pages 20–30. ACM, 2017.

[38] Ian Cassar, Adrian Francalanza, Duncan Paul Attard, Luca Aceto, and Anna Ingólfsdóttir. A generic instrumentation tool for Erlang. In Giles Reger and Klaus Havelund, editors, *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools*, volume 3 of *Kalpa Publications in Computing*, pages 48–54. EasyChair, 2017.

[39] Ian Cassar, Adrian Francalanza, Duncan Paul Attard, Luca Aceto, and Anna Ingólfsdóttir. A suite of monitoring tools for Erlang. In Giles Reger and Klaus Havelund, editors, *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools*, volume 3 of *Kalpa Publications in Computing*, pages 41–47. EasyChair, 2017.

[40] Marek Chalupa and Thomas A. Henzinger. Monitoring hyperproperties with prefix transducers. In Panagiotis Katsaros and Laura Nenzi, editors, *Runtime Verification - 23rd International Conference, RV 2023*, volume 14245 of *Lecture Notes in Computer Science*, pages 168–190. Springer, 2023.

[41] Marek Chalupa, Thomas A. Henzinger, and Ana Oliveira da Costa. Monitoring hypernode logic over infinite domains. In Bettina Könighofer and Hazem Torfah, editors, *Runtime Verification - 25th International Conference, RV 2025*, volume 16087 of *Lecture Notes in Computer Science*, pages 417–437. Springer, 2025.

[42] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018.

[43] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust - Third International Conference, POST 2014*, volume 8414 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2014.

[44] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010.

[45] Ankush Desai, Tommaso Dreossi, and Sanjit A. Seshia. Combining model checking and runtime verification for safe robotics. In Shuvendu K. Lahiri and Giles Reger, editors, *Runtime Verification - 17th International Conference, RV 2017*, volume 10548 of *Lecture Notes in Computer Science*, pages 172–189. Springer, 2017.

[46] E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 995–1072. Elsevier and MIT Press, 1990.

[47] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. RVHyper: A runtime verification tool for temporal hyperproperties. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018*, volume 10806 of *Lecture Notes in Computer Science*, pages 194–200. Springer, 2018.

[48] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. *Formal Methods Syst. Des.*, 54(3):336–363, 2019.

[49] Adrian Francalanza, Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Ian Cassar, Dario Della Monica, and Anna Ingólfsdóttir. A foundation for runtime monitoring. In Shuvendu K. Lahiri and Giles Reger, editors, *Runtime Verification - 17th International Conference, RV 2017*, volume 10548 of *Lecture Notes in Computer Science*, pages 8–29. Springer, 2017.

[50] Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. On verifying Hennessy-Milner logic with recursion at runtime. In Ezio Bartocci and Rupak Majumdar, editors, *Runtime Verification - 6th International Conference, RV 2015*, volume 9333 of *Lecture Notes in Computer Science*, pages 71–86. Springer, 2015.

[51] Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. Monitorability for the Hennessy-Milner logic with recursion. *Formal Methods Syst. Des.*, 51(1):87–116, 2017.

[52] Jan Friso Groote and Radu Mateescu. Verification of temporal properties of processes in a setting with data. In Armando Martin Haeberer, editor, *Algebraic Methodology and Software Technology, 7th International Conference, AMAST '98*, volume 1548 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 1998.

[53] Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Automata and fixpoints for asynchronous hyperproperties. *Proc. ACM Program. Lang.*, 5(POPL):1–29, 2021.

[54] Klaus Havelund and Doron Peled. Runtime verification: From propositional to first-order temporal logic. In Christian Colombo and Martin Leucker, editors, *Runtime Verification - 18th International Conference, RV 2018*, volume 11237 of *Lecture Notes in Computer Science*, pages 90–112. Springer, 2018.

[55] Klaus Havelund and Grigore Rosu, editors. *Workshop on Runtime Verification, RV 2001, in connection with CAV 2001*, volume 55(2) of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2001.

[56] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985.

[57] Thomas A. Henzinger, Konstantin Kueffner, and Emily Yu. Formal verification of neural certificates done dynamically. In Bettina Könighofer and Hazem Torfah, editors, *Runtime Verification - 25th International Conference, RV 2025*, volume 16087 of *Lecture Notes in Computer Science*, pages 54–72. Springer, 2025.

[58] Marcin Jurdzinski and Ranko Lazic. Alternation-free modal mu-calculus for data trees. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 131–140. IEEE Computer Society, 2007.

[59] Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

[60] Robert M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, 1976.

[61] Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

[62] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, 2000.

[63] Kim Guldstrand Larsen. Proof systems for satisfiability in Hennessy-Milner logic with recursion. *Theor. Comput. Sci.*, 72(2&3):265–288, 1990.

[64] K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. In Edmund M. Clarke and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16*, volume 6355 of *Lecture Notes in Computer Science*, pages 348–370. Springer, 2010.

[65] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *J. Log. Algebraic Methods Program.*, 78(5):293–303, 2009.

[66] Nicolas Markey and Philippe Schnoebelen. Model checking a path. In Roberto M. Amadio and Denis Lugiez, editors, *CONCUR 2003 - Concurrency Theory, 14th International Conference*, volume 2761 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2003.

[67] Nicolas Markey and Philippe Schnoebelen. Mu-calculus path checking. *Inf. Process. Lett.*, 97(6):225–230, 2006.

[68] Robin Milner. *Communication and Concurrency*. PHI Series in computer science. Prentice Hall, 1989.

[69] Kshirasagar Naik and Priyadarshi Tripathy. *Software Testing and Quality Assurance: Theory and Practice*. John Wiley & Sons, 2011.

[70] Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

[71] Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 37 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2013.

[72] Gordon D. Plotkin. A structural approach to operational semantics. *J. Log. Algebraic Methods Program.*, 60-61:17–139, 2004.

[73] Amir Pnueli and Aleksandr Zaks. PSL model checking and run-time verification via testers. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods, 14th International Symposium on Formal Methods*, volume 4085 of *Lecture Notes in Computer Science*, pages 573–586. Springer, 2006.

[74] César Sánchez, Gerardo Schneider, Wolfgang Ahrendt, Ezio Bartocci, Domenico Bianculli, Christian Colombo, Yliès Falcone, Adrian Francalanza, Srdan Krstic, João M. Lourenço, Dejan Nickovic, Gordon J. Pace, José Rufino, Julien Signoles, Dmitriy Traytel, and Alexander Weiss. A survey of challenges for runtime verification from advanced application domains (beyond software). *Formal Methods Syst. Des.*, 54(3):279–335, 2019.

[75] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.

[76] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. Toward verified artificial intelligence. *Commun. ACM*, 65(7):46–55, 2022.

[77] Deian Tabakov, Kristin Y. Rozier, and Moshe Y. Vardi. Optimized temporal monitors for SystemC. *Formal Methods Syst. Des.*, 41(3):236–268, 2012.

[78] Rania Taleb, Sylvain Hallé, and Raphaël Khoury. Uncertainty in runtime verification: A survey. *Comput. Sci. Rev.*, 50:100594, 2023.

[79] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5(2):285–309, 1955.

[80] Hazem Torfah, Sebastian Junges, Daniel J. Fremont, and Sanjit A. Seshia. Formal analysis of AI-based autonomy: From modeling to runtime assurance. In Lu Feng and Dana Fisman, editors, *Runtime Verification - 21st International Conference, RV 2021*, volume 12974 of *Lecture Notes in Computer Science*, pages 311–330. Springer, 2021.

The Education Column

BY

Dennis Komm and Thomas Zeume

ETH Zurich, Switzerland and Ruhr University Bochum, Germany

dennis.komm@inf.ethz.ch and thomas.zeume@rub.de

BEATCS no 148

Surveying Theory of Computing Education in the United States

Ryan E. Dougherty
Department of Electrical Engineering & Computer Science
United States Military Academy
ryan.dougherty@westpoint.edu

Tim Randolph
Department of Computer Science
Harvey Mudd College
trandolph@g.hmc.edu

Abstract

We summarize the results of a 2024 survey of theory of computing (ToC) courses that received responses from 166 institutions of higher education in the United States [8]. Although our survey is far from comprehensive, it provides new information on many aspects of ToC in U.S. bachelor's degree programs in computer science. We report on the most frequently covered topics in ToC courses as well as structural features such as typical prerequisites, section sizes, teaching modalities, and course staff.

In addition to summarizing survey results, we briefly discuss observed trends, suggest opportunities for future teaching and research, and conclude with a call for increased collaboration between American and European theoretical computer science educators.

1 Introduction

While attending the 2024 Technical Symposium on Computer Science Education (SIGCSE TS) in Portland, Oregon, a group of computer science educators convened for lunch. We discussed our shared passion for teaching *theory of computing* (ToC): roughly, the formal concepts and theoretical frameworks that unify computer science as an academic discipline, as presented to undergraduate students during their degree program. We also lamented the relative lack of computer science education literature on the subject or its pedagogy.

As the reader of this bulletin is aware, the umbrella term "theory of computing" covers a variety of related topics and its boundaries are not precisely defined. The group that met at SIGCSE TS 2024 broadly agreed on the inclusion of some topics, including formal languages, automata theory, computability, and basic concepts from complexity theory. However, we disagreed about the status of others. We taught at a wide range of institutions of higher education in the United States, including small teaching colleges, large public universities, and research-focused institutions; accordingly, our disagreements might have resulted from the institutions at which we worked. Alternatively, they might have arisen from regional, personal, or sub-disciplinary differences. Moreover, we recognized that the group was not necessarily representative of theoretical computer science educators in the U.S. Even if *we* agreed on what ToC was and how to teach it, we might not necessarily be able to speak for others.

This uncertainty presented a problem. We all shared the desire to improve our teaching, but we lacked objective information about the diversity of ToC teaching practices across the country. What topics were usually being taught in U.S. colleges and universities in a first course in ToC? How were these topics taught, by whom, and how were they organized into units, courses, and curricula? We found very little existing research on this question, so we conceived a broad survey to find out.

2 A Survey of Theory of Computing Courses

The survey project became a working group convened for the 2024 ACM Virtual Global Computing Education Conference (SIGCSE Virtual 2024); see [7] for the initial abstract. Although the group included some European researchers, the majority of members were based in the U.S. Accordingly, we narrowed the scope of our survey to the U.S. so that the project was feasible within the working group time constraints. In this respect, the mission that motivated our survey is still incomplete: we need a better understanding of how the content and rationale of ToC education differs between educational systems in the U.S., in European countries, and elsewhere.

Our first challenge was providing a definition of ToC that was simultaneously concrete enough for respondents to understand the aim of the survey, and flexible enough to capture topics and approaches to teaching we did not anticipate. We adopted a bottom-up approach by constructing a list of "ToC topics" based on the "Computational Models and Formal Languages" unit of the 2023 ACM Computer Science curricular guidelines [14]. We then grouped these topics under three broad headings and defined a theory course (for our purposes) to be any course with substantial coverage of one or more of the following:

- **Automata theory:** finite automata, regular expressions, regular languages,

the pumping lemma, context-free languages/grammars, etc.

- **Computability theory:** Turing machines, reductions, (un)decidability, etc.
- **Introductory complexity theory:** the complexity classes P and NP, NP-hardness, NP-completeness, polynomial-time reductions, etc.

Our definition does not attempt to encompass the full breadth of theory courses in the U.S., let alone worldwide. In particular we chose to exclude some closely related subfields, including algorithms and logic, from our definition.[1] Although the list was informed by group consensus and prior assumptions about what a "theory course" might look like, we asked respondents to provide additional "missing" topics; their responses largely confirmed our expectations about what an introductory theory course might include.

We distributed copies of the survey to representatives of over 1,000 institutions of higher education in the U.S. This list included a majority of all U.S. institutions that offer bachelor's degrees in computer science or a closely related field, and all of those for which we could obtain relevant contact information. Where possible, we enlisted senior faculty members with ToC teaching experience to summarize ToC teaching at their institution, defaulting to department chairs or other senior faculty members where necessary. Our 166 survey responses spanned the wide variety of U.S. institutions of higher education, from "R1" universities with very high research spending to small liberal arts colleges (SLACs) devoted to undergraduate education. From these responses, we assembled a preliminary picture of ToC teaching.

3 The Mythical Median Theory Course

As an aid to the reader's imagination, consider the "mythical median theory course": a course that represents the majority or significant plurality of our responses in qualitative dimensions and the median in quantitative dimensions. This course is "mythical" in the sense that it may not perfectly describe any course in particular. In constructing it, we have ignored correlations between the various dimensions of our data. Furthermore, we caution that even if the plurality of *courses* have some feature in common, this feature is not necessarily experienced by the plurality of *students*: for instance, one course with a very large enrollment might contain more students than many small courses put together.

[1] Our data confirmed our working assumption that American students often encounter theory of computing and algorithms in separate courses. See Luu et al. for a survey of introductory undergraduate algorithms courses [16]. For the complete topic list and further discussion of the rationale behind it, see the full report [8].

In this section we provide only rough approximations of our data. This is intentional: we wish to give necessary context for our claims without implying any particular degree of statistical exactness. For complete data on each of the features listed below, we encourage the reader to refer to the full report [8].

3.1 Frequently Covered ToC Topics

We begin with the curriculum: from our long list of possibilities, what ToC topics are actually taught? Our imposed definition of ToC limits our results to topics associated with automata, computability, and complexity theory, but within these domains some topics are more popular than others. Perhaps surprisingly, we found general agreement on a set of core ToC topics that a majority of institutions covered somewhere in their computer science degree program. We identified the following trends:

1. **Almost all responding institutions include proof techniques as part of their CS curricula.** We asked survey respondents about only one topic that did not fall under the heading of automata, computability, or complexity: techniques for writing mathematical proofs. Proof techniques are a required part of the computer science degree program for about 90 % of respondents, and most of the remaining respondents specified that this skill was covered in a regularly offered elective course. This seems to illustrate a strong shared commitment to proof-writing among our sample. (Note that if proof techniques were part of a discrete mathematics class required for computer scientists, this would still be counted using our methodology.)

2. **Almost all responding institutions cover at least one topic in automata theory, computability, and complexity.** Approximately 90 % of institutions reported that they covered at least one ToC topic in automata theory in at least one course in their computer science curriculum. The same was true for computability and for complexity theory (each considered independently).

3. **Automata theory topic coverage.** Most responding institutions cover regular languages and (non-)deterministic finite automata in their computer science curriculum. Fewer, about half, cover context-free grammars and pushdown automata, and a minority cover other topics including other automata and language classes.

4. **Computability and complexity theory topic coverage.** Coverage of ToC topics in computability theory and complexity theory was somewhat less consistent than the core topics in automata theory. Most respondents reported covering Turing machines and basic computability and complexity-theoretic

definitions (decidability, recognizability, the complexity class P, the complexity class NP, etc.) in their computer science curricula. Slim majorities covered recursively enumerable languages, mapping (many-one) and polynomial-time reductions, and the Chomsky hierarchy. Only a minority covered topics such as space complexity, Rice's Theorem, the Cook-Levin Theorem, and other computational models equivalent in power to Turing machines, such as the lambda calculus.

The names of the courses in which ToC topics appear give us further clues into the way ToC is divided up across the curriculum. For example, almost half of the courses in our sample that cover at least one topic in automata theory include the phrase "theory of computing" or the phrase "theory of computation" in their titles. These same courses often covered computability theory and complexity theory topics as well. In other words, "ToC courses" that correspond approximately to our working definition do exist at many institutions.

Many other courses covering automata theory included more specific terms such as "formal language" or "automata theory," and a few included "programming languages" or "programming language theory." Courses covering at least one computability topic were more divided, with "theory of computing" commonly included in their titles along with the more specific term "computability" and, occasionally, "complexity." Finally, although many courses that covered at least one topic in basic complexity theory were the same as those covering automata and computability, about a third were covered in courses whose name contained the term "algorithm" (presumably, courses focused on algorithm design and analysis). Relatively few courses in which students encountered our complexity theory topics contained the word "complexity" in their title, perhaps an indication that students rarely first encounter complexity theory in a course devoted primarily to the topic.

3.2 Structural Features and Teaching Modality

With the curriculum of the mythical median theory course established, we proceed to the structural features, including prerequisites, enrollment, teaching modality, and teaching staff, that define the mythical median theory course.

1. **Prerequisite courses.** The median theory course requires a course in *discrete mathematics* (about $3/4$ of the courses in our sample), and is likely to require an *introduction to programming* course and a course on *data structures* (each about $2/3$ of the courses in our sample). It appears that the median theory course occurs midway through the degree program, after students have taken a standard set of introductory courses, but does not require other special preparation.

2. **Enrollment.** The median theory course enrolls between 10 and 29 students per section. This is likely influenced by the preponderance of small colleges in our sample, but there are too many courses that offer small sections of ToC for this to be explained by small colleges alone. Courses with enrollments in the range of 10–29 students per section account for a majority of all ToC courses reported (about 3/5 of the courses in our sample), and nearly 9 out of 10 courses in our sample had typical enrollments under 40 students per section. In other words, few sections of ToC courses are large lectures.

3. **Teaching modality.** The median theory course is likely to be taught via lecture, perhaps punctuated with sessions devoted to small-group problem-solving. It is slightly more likely to be taught from a whiteboard or blackboard than via slides, although both are common. The median theory course does not use ToC-specific digital tools, with the possible exception of the formal language software JFLAP [20] (about 1/4 of the courses in our sample). The median theory course does not use other digital tools such as clickers for live polling.

4. **Teaching staff.** The median theory course is typically taught by a permanent faculty member who is also a computer scientist; few courses are taught by faculty in other disciplines or adjunct faculty members.[2] A majority of responding institutions reported that their ToC instructors focus primarily on teaching, but a substantial minority reported that their ToC instructors focus primarily on research. Similar numbers of institutions reported that their ToC courses are taught by specialists in theory of computing / algorithms and by computer scientists with other specialties.

   Almost all sections of the median theory course have few (0–2) teaching assistants or student helpers (more than 9/10 of the courses in our sample), corresponding to the relatively small median section size.

In sum, the median theory course is relatively small and relatively traditional in the way it is taught. Most ToC courses in our sample are taught by computer scientists and require prerequisite computer science courses, implying that ToC courses in the U.S. are associated much more closely with computer science as a discipline than with other disciplines such as mathematics.

[2] Part-time instructional staff.

4 Future Opportunities for Teaching and Education Research in ToC

Our "mythical median theory course" takes place in a small lecture hall and has many features in common with traditional mathematics education, including board work, slides, and a limited reliance on digital tools. This may not be surprising, but it provides the context required to discuss teaching innovations in response to internal changes and external pressures on the field of computer science.

The suggestions we offer in this section are speculative and invite future experiments, both in the formal context of computer science education research and the informal context of one's own classroom.

- **Rethinking assessment methods.** Generative Artificial Intelligence (GenAI) threatens traditional assessment methods across many academic disciplines, although it has naturally attracted special attention among computer science education researchers [21]. Students may attempt to cheat using GenAI in any class where written work is used as a proxy for learning, and this includes the many ToC classes that rely on problem sets.[3] Moreover, they may succeed: recent research has determined that GenAI models can effectively solve standard undergraduate ToC problems [5, 9, 12, 15]. Moreover, assessments that tempt students to use GenAI have additional consequences beyond unfair grading and failure to learn the material: some CS education research suggests that overreliance on GenAI can hurt students' grades and their programming ability [13].

  In response, we suggest that ToC educators experiment with assessment methods that prioritize critical analysis and effective communication of technical material, especially when these evaluations can be conducted in person by peers or instructors. Limited existing research suggests that these methods may be effective for teaching ToC [6, 22], although more study is needed.

  A drawback to interactive and in-person assessment models is that they are difficult to implement in large courses and with less experienced instructional staff. However, our survey indicates that many ToC classes occur in small sections taught by permanent faculty, so assessments of this type may be feasible. A small ToC class might even serve as a laboratory in which to test alternative assessment methods before they are adapted to larger classes in a computer science degree program.

[3] Unfortunately, our survey does not ask about the prevalence of various assessment methods in ToC classes. We view this as an important open question for future work.

- **Increasing student agency and motivation.** Our survey results indicate that for many students, their first class in ToC is also the first class in their degree program in which they apply formal mathematical methods to computer science. This may be connected to the perception among some students that ToC is particularly difficult (perhaps even "dry") [22, 23].

  This provides a second motivation for experimentation in ToC teaching. As one potential solution, we suggest trying existing pedagogical strategies that increase student engagement by focusing on agency and motivation. For example, some early research has shown effectiveness of active learning in ToC courses [11].

  A group of related teaching strategies, referred to collectively as "alternative grading", addresses motivation to learn by challenging traditional points-based grading systems [10, 18]. A significant amount of work considers the application of alternative grading methods to computer science, but relatively little considers theoretical computer science specifically. Exceptions include Weber's implementation of mastery grading in an algorithms course [24] and Randolph's experience report on participatory governance in ToC [19].

- **Multimodal learning.** Our survey found that digital tools are rarely used in ToC classrooms. Of course, this could be due to incompatibility between existing tools and the material: perhaps ToC is simply poorly suited for multimodal learning. However, we don't believe this is the case.

  Moreover, many other visualization tools exist for ToC instruction, such as JFLAP [20], TheoryViz [2], FAdo/GUItar [1], Gidayu [3], and FSM [17]; see [4] for a general overview. A substantial minority of classes indicated using JFLAP for teaching automata theory indicates an appetite for more visual learning. Given the relative abundance and availability of visualization tools, our finding that they are rarely used in ToC courses is surprising. However, the relative abundance of visualization tools contrasts with their very limited adoption; One possibility is that many ToC educators are unaware of the tools that do exist.

  Additionally, multimodal content can easily transcend the classroom in the form of online demos and videos. Online ToC courses exist, but videos are typically limited to lecture recordings; perhaps there is a niche for scripted video content.

Our suggestions—pivot assessment away from graded problem sets towards oral evaluation, focus on student motivation, and provide more visual, intuitive, and interactive learning models—combine our limited knowledge of the present landscape of ToC teaching with recommendations familiar from similar disciplines

and learning settings. However, a more daring approach might set aside what the median ToC course *is* and ask instead, "What should a ToC course *be*?"

The ToC topics covered by the median theory course are well-established and stable, especially by the fast-moving standards of computer science. In most cases, this is for good reason: concepts such as regular languages and decidability are essential to a theoretical view of computer science. However, if the curriculum becomes static, we risk fossilizing all the auxiliary concepts that come along with our preferred abstractions. For example, is spending a week wrangling over pumping lemma proofs the most effective use of student-instructor contact time in an introductory ToC class? Instructors may disagree, and the right answer may be classroom-specific. However, we believe that continually interrogating the ToC curriculum is essential for a healthy discourse around teaching practice, which in turn is useful to educators of all types.

5 An Invitation to Collaborate

Our survey of ToC in U.S. institutions was never intended as an abstract data-gathering exercise. Rather, it was a product of our desire to become better educators. We decided that an important step towards this goal was to get a better understanding of the way people teach theory of computing now; this, we hope, will allow us to make future teaching decisions with a better understanding of the way our peers teach and design research programs that are relevant to a broader audience of ToC educators.

These goals are naturally collaborative, and we believe we will continue to benefit from a greater understanding of ToC teaching outside the United States. Therefore, we conclude this article with an invitation to computer science researchers and educators: let us collaborate and learn from each other!

Acknowledgements

First and foremost, we are deeply grateful to all of our survey co-authors: Tzu-Yi Chen, Jeff Erickson, Matthew Ferland, Dennis Komm, Jonathan Liu, Timothy Ng, Ana Smaranda Sandu, Michael Shindler, Edward Talmage, and Thomas Zeume. We also thank SIGCSE Virtual and the ACM for providing the forum for this work, working group chairs Jacqueline Whalley and Juho Leinonen for their guidance throughout the development of the project, and several anonymous referees for extensive and thoughtful feedback.

The opinions in this work are solely of the authors, and do not necessarily reflect those of the U.S. Army, U.S. Army Research Labs, the U.S. Military Academy, or the Department of Defense.

References

[1] André Almeida, Marco Almeida, José Alves, Nelma Moreira, and Rogério Reis. Fado and guitar: Tools for automata manipulation and visualization. In Sebastian Maneth, editor, *Implementation and Application of Automata*, pages 65–74, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[2] Lillian Baker, Sierra Zoe Bennett-Manke, Sebastian Neumann, Ian Njuguna, and Ryan E. Dougherty. TheoryViz: A visualizer tool for theory of computing concepts. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2*, SIGCSE TS 2025, pages 1373–1374, New York, NY, USA, February 2025. Association for Computing Machinery.

[3] Tiago Cogumbreiro and Gregory Blike. Gidayu: Visualizing automaton and their computations. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, ITiCSE 2022, pages 110–116, New York, NY, USA, 2022. Association for Computing Machinery.

[4] Ian Campbell, Anthony Notaro, and Ryan E. Dougherty. Visualization tools for cs theory: an initial literature review. (accepted to SIGCSE TS 2026).

[5] Ryan E. Dougherty, Matei A. Golesteanu, and Garrett B. Vowinkel. Large language models with reasoning on theory course exams. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 2*, ITiCSE 2025, page 785, New York, NY, USA, 2025. Association for Computing Machinery.

[6] Ryan E. Dougherty. Experiences with scaffolding research projects in theory of computing courses. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2025, pages 180–186, New York, NY, USA, 2025. Association for Computing Machinery.

[7] Ryan E. Dougherty, Tim Randolph, Tzu-Yi Chen, Jeff Erickson, Matthew Ferland, Dennis Komm, Jonathan Liu, Timothy Ng, Seth Poulsen, Smaranda Sandu, Michael Shindler, Edward Talmage, and Thomas Zeume. A survey of undergraduate theory of computing curricula. In *Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 2*, SIGCSE Virtual 2024, pages 281–282, New York, NY, USA, 2024. Association for Computing Machinery.

[8] Ryan E. Dougherty, Tim Randolph, Tzu-Yi Chen, Jeff Erickson, Matthew Ferland, Dennis Komm, Jonathan Liu, Timothy Ng, Ana Smaranda Sandu, Michael Shindler, Edward Talmage, and Thomas Zeume. A survey of undergraduate theory of computation curricula in the United States. In *2024 Working Group Reports on 1st ACM Virtual Global Computing Education Conference*, SIGCSE Virtual–WGR 2024, pages 1–14, New York, NY, USA, 2025. Association for Computing Machinery.

[9] Ming Ding, Federico Soldà, Weixuan Yuan, and Rasmus Kyng. Assessing GPT performance in a proof-based university-level course under blind grading. *Bulletin of the EATCS*, 146, 2025.

[10] Joe Feldman. *Grading for Equity: What It Is, Why it Matters, and How It Can Transform Schools and Classrooms*. Corwin Press, 2023.

[11] Michael T. Grinder, Seong B. Kim, Teresa L. Lutey, Rockford J. Ross, and Kathleen F. Walsh. Loving to learn theory: active learning modules for the theory of computing. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, SIGCSE 2002, pages 371–375, New York, NY, USA, 2002. Association for Computing Machinery.

[12] Matei A. Golesteanu, Garrett B. Vowinkel, and Ryan E. Dougherty. Can ChatGPT pass a theory of computing course? In *Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 1*, SIGCSE Virtual 2024, pages 33–38, New York, NY, USA, 2024. Association for Computing Machinery.

[13] Gregor Jošt, Viktor Taneski, and Sašo Karakatič. The impact of large language models on programming education and student learning outcomes. *Applied Sciences*, 14(10):4115, 2024.

[14] Amruth N. Kumar, Rajendra K. Raj, Sherif G. Aly, Monica D. Anderson, Brett A. Becker, Richard L. Blumenthal, Eric Eaton, Susan L. Epstein, Michael Goldweber, Pankaj Jalote, Douglas Lea, Michael Oudshoorn, Marcelo Pias, Susan Reiser, Christian Servin, Rahul Simha, Titus Winters, and Qiao Xiang. *Computer Science Curricula 2023*. Association for Computing Machinery, New York, NY, USA, 2024.

[15] Nero Li, Shahar Broner, Yubin Kim, Katrina Mizuo, Elijah Sauder, Claire To, Albert Wang, Ofek Gila, and Michael Shindler. Investigating the capabilities of generative AI in solving data structures, algorithms, and computability problems. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE TS 2025, pages 659–665, New York, NY, USA, 2025. Association for Computing Machinery.

[16] Michael Luu, Matthew Ferland, Varun Nagaraj Rao, Arushi Arora, Randy Huynh, Frederick Reiber, Jennifer Wong-Ma, and Michael Shindler. What is an algorithms course? Survey results of introductory undergraduate algorithms courses in the US. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE TS 2023, pages 284–290, 2023.

[17] Marco T. Morazan and Tijana Minic. Nondeterministic to deterministic finite-state machine visualization: Implementation and evaluation. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2024, pages 262–268, New York, NY, USA, 2024. Association for Computing Machinery.

[18] Linda B Nilson, David Clark, and Robert Talbert. *Grading For Growth: A Guide to Alternative Grading Practices That Promote Authentic Learning and Student Engagement In Higher Education*. Routledge, 2023.

[19] Tim Randolph. Participatory governance in the computer science theory classroom. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE TS 2024, pages 1091–1097, New York, NY, USA, 2024. Association for Computing Machinery. event-place: Portland, OR, USA.

[20] Susan H. Rodger and Thomas W. Finley. *JFLAP: An Interactive Formal Languages and Automata Package*. Jones & Bartlett Learning, 2006.

[21] Nishat Raihan, Mohammed Latif Siddiq, Joanna CS Santos, and Marcos Zampieri. Large language models in computer science education: A systematic literature review. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, pages 938–944, 2025.

[22] Scott Sigman. Engaging students in formal language theory and theory of computation. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE 2007, pages 450–453, New York, NY, USA, 2007. Association for Computing Machinery.

[23] Florian Schmalstieg, Marko Schmellenkamp, Jakob Schwerter, and Thomas Zeume. Difficulty generating factors for context-free language construction assignments. In *Proceedings of the 2025 ACM Conference on International Computing Education Research V. 1*, ICER 2025, pages 196–209, 2025.

[24] Robbie Weber. Using alternative grading in a non-major algorithms course. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE TS 2023, pages 638–644, 2023.

Viewpoint Column

BY

Anca Muscholl and Stefan Schmid

Bordeaux University, France
and TU Berlin, Germany
anca@labri.fr and stefan.schmid@tu-berlin.de

BEATCS no 148

Discussion Panel on the Future: Highlights 2025

Wojciech Czerwiński
University of Warsaw

For several years, I have been thinking about the idea of a general discussion about the future of our area. An ideal place for such an event would be an annual meeting embracing several specialised conferences, like Highlights of Logic, Games and Automata, shortly Highlights. This event attracts computer scientists in the so-called Track B of theoretical computer science: people publishing at conferences like LICS, ICALP B, CONCUR, PODS, CAV, POPL, etc.

In 2025 we had the opportunity to organise a discussion panel at Highlights about the future research in the automata, games and logic community and this text is a report from that event. I'll write this report in a very personal style and share with you my perspective on the organisation and the panel itself, as such a personal perspective may be more interesting to read than an attempt to show things objectively.

First a few words about the Highlights conference, as maybe not everybody is familiar with this concept. It arose from the idea to have an annual meeting which brings together most of the community. In short it allows people to come to this single event, meet everyone and hear about every important result which was shown that year. The Highlights conference started in 2013 in Paris and is a very successful event ever since.

The format of the debate was rather simple: first each speaker gives a 5-10 minutes statement about her or his opinion on the future of our area. A few weeks before the panel I sent the panelists a few guiding questions: about the direction we should go as a community, criteria for distinguishing a right research question, or the relation between our discipline and the AI revolution. In the plan, after the statements there was a possible short debate between the panelists, some panel member may want to refer to what the other one said. Finally, a time for a longer discussion with the audience was planned, when everybody could express her or his statement, add a remark or ask a question. This form (statements + short discussion + remarks from the audience) is simple and does not contain a moderated debate between the panelists, which one could expect. I made such a decision

based on the observation that it is often extremely challenging to moderate a debate between just a few panelists, make it interesting and not chaotic. Instead, eliminating moderated discussion and just letting people express their thoughts in an unmoderated framework seems better in my opinion. Anyway it is usually the most common form of debate in the real world. For all of that we had 90 minutes.

When choosing the panelists I took care about certain diversity among them in a few dimensions (like stage of career or subfields of our area), but my main purpose was to find people who enjoy long discussions for their own sake, sharing reflections and philosophy. I'm very grateful to Joël Ouaknine, Sophie Pinchinat, Szymon Toruńczyk and Georg Zetzsche, who kindly accepted my invitation. I'd like to particularly thank Szymon with whom I've discussed already in advance, in Warsaw many possible aspects of this debate and with whom I casually like to discuss topics like foundations of mathematics. I'm also very grateful to Christel Baier, who was PC Chair of Highlights this year and who constantly helped me on different stages of the preparation of this event.

During the debate a lot of various topics were touched upon and it would take very long time to describe all the thoughts brought by the panelists and by the audience. Therefore, I'll mainly focus on the ideas, which particularly resonated with me personally. The first statement was given by Szymon Toruńczyk. I particularly liked his opinion on what should guide us to find valuable research topics. Szymon emphasised that this should be mathematical beauty and depth and also interactions with other fields of mathematics. Even though it might seem quite a strong statement, I particularly agree with his opinion about the future with respect to the interaction with artificial intelligence: "There will be an increasing number of papers which claim relevance for AI, but I think in our community, we should always evaluate the mathematical depth and beauty, and not the claim of practicality or impact on other areas. I think this should be judged by experts on the other areas.". Szymon also expressed a few thoughts about interaction of verification and AI. I found interesting his prediction that the area of verification may get very useful to verify what AI will produce, i.e. to get sure that the proofs, code, or other texts produced by AI are not hallucination or some erroneous or mistaken statements.

The next speaker was Joël Ouaknine and he has formulated a very interesting thought about open problems in our area. He emphasised that having flagship open problems helps very much a community. Researchers working in algorithmics or complexity theory have their famous problems like for example P vs NP or improving the exponent of matrix multiplication. Such problems guide the research and increase visibility outside of the small area. Joël postulated that it would be very valuable to establish some list of important open problems from our community. This thread was later continued during the discussion. I liked the idea very much and have asked how can we create such a list, which is pro-

posed by Joël. Should we have a new Hilbert, who proposes a set of fundamental problems, or maybe should we establish some committee, which will choose the problems? In response to that Georg Zetzsche answered that maybe we should create on our own homepages lists of favourite open problems. I like this idea, I have such a list on my homepage, but I personally think this way of promoting open problems is too weak. Joël responded by a reference to the list of Millennium Prize Problems. He said that similarly as it was done with Millenium Prize Problems it would be good to create a committee consisting of researchers representing various subfields in order to propose a list of open problems. Moreover, it would be encouraging to fund a prize, which would be given for solving any problem from the list. To me personally this idea is the most interesting result of the debate and I think it would be absolutely fantastic if such a list could appear in next few years.

The third speaker was Georg Zetzsche. Among other ideas he noted that in our community there is an immense emphasis on algorithms. Georg postulated that we should not only focus on the algorithmic aspects of automata, but it is also very important to seek for connections between various notions, like for example the equivalence MSO = Reg. In relation to this remark he recommended to read an essay "The Two Cultures of Mathematics" by Timothy Gowers, a Fields medalist, which discusses the ideas of problem-solvers and theory-builders. I found this observation about our community very interesting and worth reflection, that's why I mention Georg's remark and reading recommendation.

As the last panel speaker Sophie Pinchinat was describing her point of view. She focused mainly on the need for higher recognition of our area in the society. I particularly liked one observation by her. Sophie noticed that probably the general public does not recognise that there is some basic research within computer science. Indeed, it is widely known today that computer science is one of the most important technologies of our times and probably most of the people are convinced that artificial intelligence changes the world. However, I'm not sure how many of these people notice deep connections of computer science to theory, and to math in particular.

After these statements of panel speakers there was a short discussion between the panel members and longer sequence of remarks and observations from the audience. Several people shared their thoughts, I will focus only on three of those, which particularly resonated with me.

The first remark, which seems important to me, was a question by Marcin Jurdziński. He said that as a community we should ask ourselves a question: "What is our identity?". Do we see ourselves as pure mathematicians? Do we see us interacting and cooperating with engineers? What is our ambition and profile of which person we would like to fit? Marcin mentioned here a few names of prominent researchers, he said: "Are we Euler? Are we Turing? Are we Kanellakis?

Are we Shelah? Are we Wiles? I don't know, who are we?". In my opinion this question is fundamental in our situation, of community situated somehow at the border of several paradigms of doing science. Even people sitting in the same office and co-authoring many papers may have different philosophical approaches to that issue. I've later talked in person with Marcin and he emphasised that his perspective on this question is very pragmatic. It is not only a philosophical question. The answer to it implies our actions.

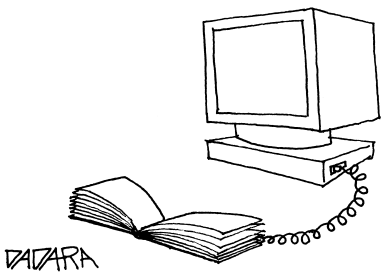
Another high level thought was formulated by Lorenzo Clemente. He noticed that in the papers we write we often formulate our results in the form: problem X has complexity Y. However, when we talk with our colleagues about our recent research we rather share an interesting lemma we have proven or a novel technique, which we have discovered. A conclusion is: maybe we should formulate differently our results? I think this is a very interesting idea. I'll take this opportunity to share my personal opinion on a similar matter. Indeed, we very often study questions of the form: does problem X have complexity Y. However, why do we do that? In my opinion the right answer to it should be that these questions are not our goals by itself, these are the questions which guide our research into the right direction. Our true goal in fundamental research should be finding new techniques, new natural notions, new deep connections between mathematical structures. However, it is hard to ask ourselves: find a new technique. Instead, we need more concrete, measurable goals, which guide us towards search for right notions. In my opinion this is exactly the role of the algorithmic questions we ask. They are not the goal, they are the path. Coming back to Lorenzo's question: how should we formulate them? I don't think there is an easy answer to it. Algorithmic formulation is probably easier to appreciate for people a bit further from the problem. Imagine you solved P vs NP problem by a novel, insightful technique. The technique is the core of the new understanding, but you need to say that it solved P vs NP question. Otherwise only a tiny fraction of the community will appreciate this breakthrough. However, at the same time, deep understanding was done by the technique and you will explain the technique to your colleagues. I don't know the answer to Lorenzo's question, but I really appreciate it.

A final remark, which I'd like to share was formulated by Thomas Colcombet. He asked how the culture of publishing on conferences influences our research. I've heard many times discussions about a nonstandard model of conferences in the community of theoretical computer science, but never this particular question - how does it change our way of performing science. I think this is also a question worth reflection.

Summarising, I'm happy that such a discussion could take place. Personally, I find the idea of creating a list of flagship open problems in our area the most inspiring thought from the debate. I hope that one day there will be a right momentum and such a list, together with a prize, will be created and funded. On top

of that, I hope that our meeting may inspire more panel discussions on various events in our community, either on similar topics, or very different ones. In any way, I deeply believe it is important to always simultaneously with the actions think where are we heading and why there.

News and Conference Reports



Book Announcement: "Proof Theory and Logic Programming: Computation as Proof Search"

Dale Miller

Cambridge University Press has recently published my book, "Proof Theory and Logic Programming: Computation as Proof Search" (December 2025). A preprint of the full text is available for download [1].

While the **proof-as-program** approach—where computation is driven by proof normalization—is well-documented in the literature, this book explores an alternative formalization: the **proof-search** paradigm. Often identified with logic or relational programming, proof search is traditionally introduced through resolution and Horn clauses. This work, however, anchors the paradigm in structural proof theory and the sequent calculus, reframing computation as the systematic search for cut-free proofs. By treating computation as a goal-directed process, this book bridges the gap between high-level logical specifications and their execution.

1 Key Themes and Coverage

This book establishes the sequent calculus as a robust framework for describing proof search, with a rigorous focus on structural rules, focused proofs, and cut-elimination theorems. By tracing the transition from classical to intuitionistic and linear logic, it demonstrates how these systems expand the expressive power of logic programming. Using the lens of focusing, the text develops the logical foundations for Prolog, λProlog, and two distinct linear logic programming languages. Furthermore, the framework accommodates both first-order and higher-order quantification, with the completeness of focused proofs proven via detailed cut-elimination arguments.

2 From Theory to Application

Beyond theoretical foundations, the book offers numerous examples of applying proof theory to reason about logic programs. It features several chapter-length

case studies that apply these principles to modern computer science problems:

- **Encoding Security Protocols**: Modeling communication and cryptographic properties.
- **Formalizing Operational Semantics**: Using logic to specify and analyze the behavior of programming languages.
- **Static Analysis**: Performing collection analysis and formal reasoning for Horn clauses.

A central question addressed is: "Why should one incorporate higher-order quantification or linear logic into a logic program?" The application-oriented chapters provide several examples that leverage these rich logical specifications.

3 Audience

Whether you are a graduate student in logic and computation, a researcher in programming language design, or an educator seeking a principled textbook on logic programming, this book serves as a comprehensive resource. It assumes minimal prerequisites, making it accessible to newcomers while offering insights for seasoned experts. The text contains over 90 exercises, and many are given full or partial solutions. "Proof Theory and Logic Programming" distills forty years of research into a clear, principled journey from theory to design and application. It invites the computer science community to reconsider how logic can specify computation, and encourages the proof theory community to explore new dynamics and applications of the sequent calculus.

For more information, including the table of contents and endorsements, please visit the book's webpage at the link provided below.

References

[1] Dale Miller. *Proof Theory and Logic Programming: Computation as Proof Search*. Cambridge University Press, 2025. DOI: 10.1017/9781009561280 Preprint available at `https://www.lix.polytechnique.fr/Labo/Dale.Miller/ptlp/`.

Report on Yurifest 2025

Festchrift Conference in Honour of Yuri Gurevich

Guillermo Badia & Martin Wirsing
University of Queensland & Ludwig-Maximilians-Universität München

YURIFEST 2025, a conference aimed to honor the 85th anniversary of Prof Yuri Gurevich, and his outstanding contributions towards advancing logic and computer science took place in Munich from 20 June to 22 June, 2025. The conference was organized by Guillermo Badia (University of Queensland, Australia) and Martin Wirsing (Ludwig-Maximilians-Universität München).

Gurevich's interests have spanned a broad spectrum of logic and computer science, including decision procedures, the monadic theory of order, abstract state machines, formal methods, foundations of computer science, privacy and security, quantum computing, and much more. Many of these areas were reflected in the topics of the symposium.

A broad range of topics in logic and computer science was covered by the following invited talks:

1. ***Etienne Grandjean*** (University of Caen, France). "How does preprocessing make it possible to obtain constant time?"
2. ***Alexander Shen*** (CNRS, France). "All Kolmogorov complexity functions are optimal, but are some more optimal?"
3. ***Nachum Dershowitz*** (Tel Aviv University, Israel). "Graphical Operational Semantics"
4. ***Wolfgang Reisig*** (Humboldt-Universität Berlin, Germany). "Reflections about Yuri's Abstract State Machines"
5. ***Philip Hieronymi*** (Bonn University, Germany). "A strong version of Cobham's theorem"
6. ***Vladimir Lifschitz*** (University of Texas, Austin, USA). "Verifying equivalence of declarative programs"
7. ***Andrei A. Bulatov*** (Simon Fraser University, Canada). "Commutative vs non-commutative CSP"
8. ***Janos Makowsky*** (Technion, Israel). "Supercongruences for Integer Sequences."
9. ***Arnon Avron*** (Tel Aviv University, Israel). "Predicative Set Theory and Predicative Analysis"

10. ***Valentin Goranko*** (Stockholm University, Sweden). "Almost sure theories of fragments of monadic second-order logic"
11. ***Sven Manthe*** (Bonn University, Germany). "The Borel monadic theory of order is decidable"
12. ***Alexander Rabinovitch*** (Tel Aviv University, Israel). "Rabin Uniformization Problem with Restricted Domain Variables"
13. ***Andreas Blass*** (University of Michigan, USA). "Infinite games, realistic games, and game semantics"
14. ***Manfred Droste*** (Universität Leipzig, Germany). "Random constructions imply symmetry"
15. ***Ronald de Wolf*** (CWI, Amsterdam, The Netherlands). "Quantum Proofs for Classical Theorems"
16. ***Bertrand Meyer*** (Constructor Institute of Technology, Switzerland). "A new PRISM Programming Really Is Simple Mathematics"
17. ***Stefan Schmid*** (Technical University, Berlin, Germany). "Self-adjusting Topologies"
18. ***Yuri Gurevich*** (University of Michigan, USA). "The nature of nondeterministic (probabilistic, quantum, etc.) algorithms"

Details of the program and recordings of talks can be found on the website `https://sites.google.com/view/yurifest2025/home?authuser=0`.

There was an informal get together on 19 June at Beergarden, Wirtshaus and a conference dinner on 21 June at Augustiner am Dom.

Report on CIAA 2025

29th Conference on Implementation and Application of Automata Palermo, Italy, September 22–25, 2025

Markus Holzer

The 29th International Conference on Implementation and Application of Automata (CIAA) was held at Sala Lanza of the University of Palermo's Botanical Garden on 22–25 September 2025. It is worth mentioning that this was the first time CIAA was hosted in Italy. The conference was attended by 50 participants from 13 countries.

It was organized by the Universitá degli Studi di Palermo under the direction of Giuseppa Castiglione and Sabrina Mantaci. In addition to 22 accepted submissions, there were three invited talks, namely by Marie-Pierre Béal on *Symbolic dynamics and automata*, Dirk Nowotka on *On Decision Problems of Pattern Languages*, and Alberto Policriti on *Wheeler languages and automata*. The proceedings were published as volume 15981 of Lecture Notes in Computer Science (LNCS) from Springer-Verlag. A special issue of selected contributions is planned in the International Journal of Foundations of Computer Science (IJFCS). The invitation to this special issue has been send out. The award for the best paper was given to *Engineering an LTLf Synthesis Tool* by Alexandre Duret-Lutz, Shufang Zhu, Nir Piterman, Giuseppe De Giacomo, and Moshe Vardi—congratulations to the authors.

A welcome cocktail party was organized at the first conference day in the evening at the Botanical garden. The Botanical Garden of the University of Palermo ranks among the most significant academic institutions in Italy. For more than 230 years, this vast open-air museum has fostered the study and dissemination of countless plant species across Sicily, Europe, and the Mediterranean region, including many originating from tropical and sub-tropical areas. In addition, coffee breaks and lunches were also complimentary, allowing the participants to stay together and worry only about scientific matters. After a guided tour through Palermo, the vibrant capital of Sicily, the conference banquet took place at the restaurant La locanda del gusto, with fine exquisite cuisine. Palermo is a city where centuries of history, art, and culture intertwine—from its Arab-Norman architecture to its bustling markets and grand baroque churches—and where Emperor Frederick II once fostered a remarkable era of intellectual and cultural flourishing. Surrounded by mountains and the sea, it captivates visitors with its golden light, rich cuisine, and timeless Mediterranean charm. The weather was fine, except for one day, where it was raining during the morning session, and the atmosphere was very friendly.

We thank all the people who made it possible for CIAA 2025 to be such a success. CIAA 2025 was sponsored by: Dipartimento di Matematica e Informatica @ Unipa, Universitá degli Studi di Palermo, and Springer-Verlag. During the business meeting it was announced that next years' CIAA will take place in Kingston, Ontario, Canada, from August 5 to 8, 2026.

Obituary for Gilles Dowek

Serge Abiteboul Pablo Arrighi Nachum Dershowitz
Gérard Huet Jean-Pierre Jouannaud Claude Kirchner

March 4, 2026

The theoretical computer science community is heartbroken by the untimely loss of Gilles Dowek, who passed away on 21 July 21 2025, in Paris, at the age of 58. His husband, sister, brother, family, many friends and colleagues were with him during his final days.

Gilles Dowek was born on 20 December 1966 in Paris. A brilliant student, he entered École Polytechnique, nicknamed X, in 1985. In 1991, he defended a doctoral thesis, under the supervision of Gérard Huet, at the University of Paris Diderot [1]. He started his career as an Inria researcher in 1993. Invited to apply as a professor by Jean-Pierre Jouannaud, director of LIX, the Laboratory for Informatics at X, and by Gilles Kahn, member of the recruiting committee, he was appointed Professor there in 2002, the youngest professor of the École at that time. He taught from 2002 to 2010, and was a main asset in making computer science popular among students. *La Jaune et la Rouge*, a newsletter by and for the alumni, pointed out that the École would have been wise to attract him more permanently. In 2010, he returned to Inria, becoming at the same time a professor at the École normale supérieure Paris-Saclay and a member of the Laboratory for Formal Methods (LMF), until his death. Indeed, Gilles Dowek loved teaching, and students loved him as a professor, be they sophomore or graduate.

Pedagogy

An example of his incredible ability to make seem obvious what is not is a striking introductory lecture to programming [2]. If you read French, read it! His interest in pedagogy led him and Nachum Dershowitz to describe the advantages of a two-dimensional Turing machine for ease and clarity of programming [3]. A universal machine could be fully described by an easily understandable 2D program taking up half a page. He programmed an interpreter in OCaml for such a device, and drafted a children's book based on it, laying down the principles of computation.

Awards

His passion for programming began early: At the age of fifteen, he modeled the Mastermind game in order to optimize queries, which earned him the Philips Scientific Prize for Young People in 1982, and the European Philips Contest for Young Researchers and Inventors in 1983. These were the first of his many awards. He won the Prix d'Alembert for high school students from the French Mathematical Society in 2000. In the field of computer science and applied mathematics, he received the Inria–Académie des Sciences Grand Prize in 2023. In the field of philosophy and the humanities, he received the Grand Prix de Philosophie from the Académie Française in 2007 for his book *Les Métamorphoses du calcul. Une étonnante histoire de mathématiques* (The Metamorphoses of Calculation: An Astonishing History of Mathematics). In this book, he shows how mathematics and logic were transformed in the 20th century with the integration of the concept of computation. In 2024, he received the Médaille Histoire des Sciences et Épistémologie from the Académie des Sciences. These are all signs that he was one of the foremost scientists in his field, but also much more than that, as we shall see.

Science popularization

Gilles shared with his friend, the famous philosopher Michel Serres, a taste for vivid illustrations and accurate storytelling. He excelled at making the abstract clear and striking people's minds with relevant, often unexpected examples. Not surprisingly, he was thus a great communicator of science. He was the author, among other chronicles, of the column "Homo sapiens informaticus" in the magazine *Pour la science* from 2014 to 2021. He also contributed many articles to the "Blog binaire" of *Le Monde* newspaper. He coauthored the play, *Le temps des algorithmes* (The Age of Algorithms), with Serge Abiteboul, which won the La science se livre Award in 2018. To engage the general public to envision the future with AI, Gilles Dowek also co-authored "Qui a hacké Garoutzia?" (Who Hacked Garoutzia?) with Serge Abiteboul and Laurence Devillers, which premiered at the Avignon Off Festival in July 2023, directed by Lisa Bretzner.

Education

Gilles Dowek has had a decisive influence on the development of computer science education in France. He contributed to the Academy of Sciences' influential report "L'enseignement de l'informatique en France, Il est urgent de ne plus attendre" (Computer science education in France: We can't wait any longer), in 2013. He helped develop the official French CS curriculum. He contributed to the effective implementation in France of open and widely available online courses (MOOCs), which continue today in *Fun* (France Université Numérique) and Inria's Learning Lab. He wrote with coauthors textbooks such as *Informatique et sciences du numérique: Spécialité ISN en terminale S*, a popular computer science textbook for high school. Beyond science, from April 2023 on, he was a member of the Higher Council for Education Programs.

Ethics and human rights

Since the early 2000s, Gilles became involved in promoting thinking about the societal impact of digital science and technology. He participated in CERNA (Commission for Reflection on Ethics in Digital Science and Technology Research) and was a particularly active and valued member of CNPEN (National Steering Committee on Digital Ethics), where his analyses and proposals were remarkably insightful. Quoting the site of that committee: "To those who had the opportunity to work alongside him, he leaves behind a valuable example of critical and responsible thinking that was both inventive and structured, guided by a constant concern for justice, inclusivity, and clarity; thinking driven by a constant desire to empower everyone, starting with the youngest members of society." More recently, he participated in the French Digital Council. All his life, Gilles was engaged in the defense of humanistic values. In particular, he chaired the Association for the Recognition of the Rights of Homosexual and Trans People to Immigration and Residence.

Research contributions

As can be expected, Gilles Dowek contributed to many different areas, spanning several, seemingly distant scientific fields and topics, all having a relationship to mathematical logic.

Typed lambda calculi. His first research works focused on algorithms for typed lambda calculi, in order, on the one hand, to be able to represent terms and propositions of higher-order logic, and, on the other hand, to express their proofs in a Gentzen-like natural deduction style, in the spirit of the Curry-Howard correspondence. His PhD thesis tackles the question of automating proofs in the Calculus of Constructions, yielding a higher-order generalization of Nicolas de Bruijn work on Automath. This early work contributed to the design of the proof system elaborated for the Calculus of Constructions carried out within Projet Formel at Inria Rocquencourt, which ultimately gave birth to

the proof assistant Coq. In particular, he proposed a uniform way of building proofs based on unification [4].

Subsequently, Gilles developed a keen interest in matching and unification algorithms in various typed lambda calculi, in order to design semi-automatic proof techniques. In particular, he showed that third-order matching is decidable, improving previous work of Gérard Huet [5], and completing his analysis of second-order matching in the different systems of Henk Barendregt's cube of typed lambda-calculi [6]. On the other hand, he showed that the third-order case was undecidable in the case of dependent types, an ingredient essential for representing predicates [7,8]. In collaboration with Thérèse Hardin, Claude Kirchner and Frank Pfenning, he also showed that unification could be analyzed in detail thanks to an explicit substitution calculus [9]. Finally, he was invited to write the "Matching and Unification" chapter for the *Handbook of Automated Reasoning* [10].

Deduction modulo. Mathematical reasoning involves deduction steps as well as computational steps, which can be seen as a way to search within a congruence class of expressions one that better fits a specific deduction step. How deduction and computation interact when operating on terms or on propositions was an important general research theme which Gilles contributed to in collaboration with Claude Kirchner and Thérèse Hardin [11], which lead to the general theory of deduction modulo [12].

Dedukti and Logipedia. While usual computer languages all allow the same functions to be expressed, namely, the computable algorithms, the same is not true for the expression of proofs. Proof expression languages are often incomparable. Gilles therefore set himself an inspiring task: to design a universal proof language capable of representing proofs from various logical systems and facilitating their mutual translation, thereby providing support for an ambitious project, a universal library of formal mathematical proofs. To this end, he showed with his students that a lambda calculus equipped with both dependent types and user-defined rewrite rules was a good candidate for a universal proof language [13], one that he named Dedukti. An implementation of Dedukti became therefore the kernel of this endeavor, and was later completed by various translation tools, from and to, other languages such as Matita, Coq, Agda, HOL, HOL-Light, and more when new languages emerged, such as Lean. A major theoretical question that popped up is whether proofs in Dedukti make sense, requiring that the user-defined rewrite rules are confluent (diverging computations can always be joined). This classical problem became an important trend in the team and culminated with a very general theorem for the case where rewrite rules are left-linear [14], the case of non-linear ones being also studied by his students and collaborators. Then, his dream came true with Logipedia [15].

Foundations and physics. In addition to all that, Gilles Dowek also developed an interest in various relationships between computer science and physics. In particular, he was keen to clarify that there is no incompatibility between the continuous and chaotic nature of physics and the discrete nature of computability [16]. He even argued that the opposite is true: some physics postulates entail the Church-Turing thesis. This was first addressed by Robin Gandy, Alan Turing's PhD student, who proved that the thesis was a consequence of postulates about classical physics, such as a bound on the density of information. With Pablo Arrighi, Gilles generalized this proof account for quantum physics [17]. Together, they also designed the first quantum programming language allowing for quantum superpositions of programs [18]. They also came up with a first formalized explanation of the arrow of time without the need to assume a low entropy initial state [19]. And with Alejandro Diaz-Caro, he developed a novel logical connective, capturing the peculiar form of non-determinism that is inherent to quantum mechanics [20], hence laying grounds for the development of proofs of quantum programs.

In short

Gilles Dowek was a computer scientist, logician, and philosopher. In all aspects of his life, Gilles was brilliant, original, funny, with an enlightening and sometimes irreverent sense of humor. He was rigorous, passionate, warm, and above all, deeply humanistic. He will be terribly missed.

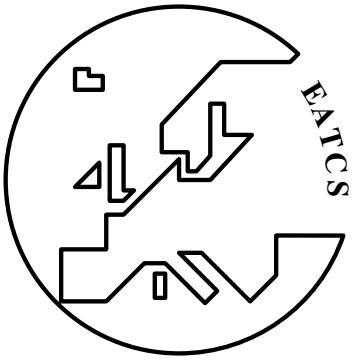
References

[1] Gilles Dowek: Démonstration Automatique dans le Calcul des Constructions (Automated Theorem Proving in the Calculus of Constructions.) Paris Diderot University, France, 1991.
[2] Gilles Dowek: Le langage mathématique et les langages de programmation. https://inria.hal.science/hal-04045740.
[3] Nachum Dershowitz, Gilles Dowek: Universality in Two Dimensions. J. Log. Comput. 26(1): 143–167 (2016).
[4] Gilles Dowek: A Complete Proof Synthesis Method for the Cube of Type Systems. J. Log. Comput. 3(3): 287–315 (1993).
[5] Gilles Dowek: Third Order Matching is Decidable. Ann. Pure Appl. Log. 69(2–3): 135–155 (1994).
[6] Gilles Dowek: A Second-Order Pattern Matching Algorithm for the Cube of Typed Lambda-Calculi. MFCS 1991: 151–160 (1991).
[7] Gilles Dowek: The Undecidability of Typability in the Lambda-Pi-Calculus. TLCA 1993: 139–145 (1993).
[8] Gilles Dowek: The Undecidability of Pattern Matching in Calculi Where Primitive Recursive Functions are Representable. Theor. Comput. Sci. 107(2): 349–356 (1993).
[9] Gilles Dowek, Thérèse Hardin, Claude Kirchner, Frank Pfenning: Unification via Explicit Substitutions: The Case of Higher-Order Patterns. JICSLP 1996:

259–273 (1996).
[10] Gilles Dowek: Higher-Order Unification and Matching. Handbook of Automated Reasoning: 1009–1062 (2001).
[11] Gilles Dowek, Thérèse Hardin, Claude Kirchner: Theorem Proving Modulo. J. Autom. Reason. 31(1): 33–72 (2003).
[12] Gilles Dowek: Deduction Modulo Theory, in Proc. RTA-TLCA, Lecture Notes in Computer Science 8560, Springer, (2014).
[13] Denis Cousineau, Gilles Dowek: Embedding Pure Type Systems in the Lambda-Pi-Calculus Modulo. TLCA 2007: 102–117 (2007).
[14] Gilles Dowek, Gaspard Férey, Jean-Pierre Jouannaud, Jiaxiang Liu: Confluence of Left-Linear Higher-Order Rewrite Theories by Checking their Nested Critical Pairs. Math. Struct. Comput. Sci. 32(7): 898–933 (2022).
[15] Gilles Dowek, François Thiré: Logipedia: A Multi-system Encyclopedia of Formal Proofs. CoRR abs/2305.00064 (2023).
[16] G. Dowek: La forme physique de la thèse de Church et la sensibilité aux conditions initiales. (The Physical Church Thesis and the Sensitivity to Initial Conditions.) In Samuel Tronçon et Jean–Baptiste Joinet, Ouvrir la logique au monde, Hermann, 2009.
[17] Pablo Arrighi, Gilles Dowek: The Physical Church-Turing Thesis and the Principles of Quantum Theory. CoRR abs/1102.1612, Int. J. Found. of Computer Science, 23(5), 1131–1145, (2012).
[18] Pablo Arrighi, Gilles Dowek: A Linear-Algebraic Lambda Calculus: Higher-Order and Confluence, Proceedings of RTA'08, Hagenberg, July 2008. LNCS 5117, 17, (2008).
[19] Pablo Arrighi, Gilles Dowek, Amélia Durbec: A Time Arrow Without Past Hypothesis: A Toy Model Explanation. New J. Phys. 26 113019, (2024).
[20] Alejandro Díaz-Caro, Gilles Dowek: A New Connective in Natural Deduction, and its Application to Quantum Computing, Theoretical Computer Science, vol. 957, 113840, (2023).

The Bulletin of the EATCS

European Association for Theoretical Computer Science



E A T C S

201

EATCS

HISTORY AND ORGANIZATION

EATCS is an international organization founded in 1972. Its aim is to facilitate the exchange of ideas and results among theoretical computer scientists as well as to stimulate cooperation between the theoretical and the practical community in computer science.

Its activities are coordinated by the Council of EATCS, which elects a President, Vice Presidents, and a Treasurer. Policy guidelines are determined by the Council and the General Assembly of EATCS. This assembly is scheduled to take place during the annual **I**nternational **C**olloquium on **A**utomata, **L**anguages and **P**rogramming (ICALP), the conference of EATCS.

MAJOR ACTIVITIES OF EATCS

- Organization of ICALP;
- Publication of the "Bulletin of the EATCS;"
- Award of research and academic career prizes, including the EATCS Award, the Gödel Prize (with SIGACT), the Presburger Award, the EATCS Distinguished Dissertation Award, the Nerode Prize (joint with IPEC) and best papers awards at several top conferences;
- Active involvement in publications generally within theoretical computer science.

Other activities of EATCS include the sponsorship or the cooperation in the organization of various more specialized meetings in theoretical computer science. Among such meetings are: CIAC (Conference of Algorithms and Complexity), CiE (Conference of Computer Science Models of Computation in Context), DISC (International Symposium on Distributed Computing), DLT (International Conference on Developments in Language Theory), ESA (European Symposium on Algorithms), ETAPS (The European Joint Conferences on Theory and Practice of Software), LICS (Logic in Computer Science), MFCS (Mathematical Foundations of Computer Science), WADS (Algorithms and Data Structures Symposium), WoLLIC (Workshop on Logic, Language, Information and Computation), WORDS (International Conference on Words).

Benefits offered by EATCS include:

- Subscription to the "Bulletin of the EATCS;"
- Access to the Springer Reading Room;
- Reduced registration fees at various conferences;
- Reciprocity agreements with other organizations;
- 25% discount when purchasing ICALP proceedings;
- 25% discount in purchasing books from "EATCS Monographs" and "EATCS Texts;"
- Discount (about 70%) per individual annual subscription to "Theoretical Computer Science;"
- Discount (about 70%) per individual annual subscription to "Fundamenta Informaticae."

Benefits offered by EATCS to Young Researchers also include:

- Database for Phd/MSc thesis
- Job search/announcements at Young Researchers area

(1) THE ICALP CONFERENCE

ICALP is an international conference covering all aspects of theoretical computer science and now customarily taking place during the second or third week of July. Typical topics discussed during recent ICALP conferences are: computability, automata theory, formal language theory, analysis of algorithms, computational complexity, mathematical aspects of programming language definition, logic and semantics of programming languages, foundations of logic programming, theorem proving, software specification, computational geometry, data types and data structures, theory of data bases and knowledge based systems, data security, cryptography, VLSI structures, parallel and distributed computing, models of concurrency and robotics.

SITES OF ICALP MEETINGS:

- Paris, France 1972
- Saarbrücken, Germany 1974
- Edinburgh, UK 1976
- Turku, Finland 1977
- Udine, Italy 1978
- Graz, Austria 1979
- Noordwijkerhout, The Netherlands 1980
- Haifa, Israel 1981
- Aarhus, Denmark 1982
- Barcelona, Spain 1983
- Antwerp, Belgium 1984
- Nafplion, Greece 1985
- Rennes, France 1986
- Karlsruhe, Germany 1987
- Tampere, Finland 1988
- Stresa, Italy 1989
- Warwick, UK 1990
- Madrid, Spain 1991
- Wien, Austria 1992
- Lund, Sweden 1993
- Jerusalem, Israel 1994
- Szeged, Hungary 1995
- Paderborn, Germany 1996
- Bologne, Italy 1997
- Aalborg, Denmark 1998
- Prague, Czech Republic 1999
- Genève, Switzerland 2000
- Heraklion, Greece 2001
- Malaga, Spain 2002
- Eindhoven, The Netherlands 2003
- Turku, Finland 2004
- Lisabon, Portugal 2005
- Venezia, Italy 2006
- Wrocław, Poland 2007
- Reykjavik, Iceland 2008
- Rhodes, Greece 2009
- Bordeaux, France 2010
- Zürich, Switzerland 2011
- Warwick, UK 2012
- Riga, Latvia 2013
- Copenhagen, Denmark 2014
- Kyoto, Japan 2015
- Rome, Italy 2016
- Warsaw, Poland 2017
- Prague, Czech Republic 2018
- Patras, Greece 2019
- Saarbrücken, Germany (virtual conference) 2020
- Glasgow, UK (virtual conference) 2021
- Paris, France 2022
- Paderborn, Germany 2023
- Tallinn, Estonia 2024

(2) THE BULLETIN OF THE EATCS

Three issues of the Bulletin are published annually, in February, June and October respectively. The Bulletin is a medium for *rapid* publication and wide distribution of material such as:

- EATCS matters;
- Technical contributions;
- Columns;
- Surveys and tutorials;
- Reports on conferences;
- Information about the current ICALP;
- Reports on computer science departments and institutes;
- Open problems and solutions;
- Abstracts of Ph.D. theses;
- Entertainments and pictures related to computer science.

Contributions to any of the above areas are solicited, in electronic form only according to formats, deadlines and submissions procedures illustrated at `http://www.eatcs.org/bulletin`. Questions and proposals can be addressed to the Editor by email at `bulletin@eatcs.org`.

(3) OTHER PUBLICATIONS

EATCS has played a major role in establishing what today are some of the most prestigious publication within theoretical computer science.

These include the *EATCS Texts* and the *EATCS Monographs* published by Springer-Verlag and launched during ICALP in 1984. The Springer series include *monographs* covering all areas of theoretical computer science, and aimed at the research community and graduate students, as well as *texts* intended mostly for the graduate level, where an undergraduate background in computer science is typically assumed.

Updated information about the series can be obtained from the publisher.

The editors of the EATCS Monographs and Texts are now M. Henzinger (Vienna), J. Hromkovič (Zürich), M. Nielsen (Aarhus), G. Rozenberg (Leiden), A. Salomaa (Turku). Potential authors should contact one of the editors.

EATCS members can purchase books from the series with 25% discount. Order should be sent to:

Prof.Dr. G. Rozenberg, LIACS, University of Leiden,
P.O. Box 9512, 2300 RA Leiden, The Netherlands

who acknowledges EATCS membership and forwards the order to Springer-Verlag.

The journal *Theoretical Computer Science*, founded in 1975 on the initiative of EATCS, is published by Elsevier Science Publishers. Its contents are mathematical and abstract in spirit, but it derives its motivation from practical and everyday computation. Its aim is to understand the nature of computation and, as a consequence of this understanding, provide more efficient methodologies.

The Editor-in-Chief of the journal currently are D. Sannella (Edinburgh), L. Kari and P.G. Spirakis (Patras).

ADDITIONAL EATCS INFORMATION

For further information please visit `http://www.eatcs.org`, or contact the President of EATCS:

Prof. Giuseppe F. Italiano,
Email: `president@eatcs.org`

EATCS MEMBERSHIP

DUES

The dues are € 40 for a period of one year (two years for students / Young Researchers). Young Researchers, after paying, have to contact `secretary@eatcs.org`, in order to get additional years. A new membership starts upon registration of the payment. Memberships can always be prolonged for one or more years.

In order to encourage double registration, we are offering a discount for SIGACT members, who can join EATCS for € 35 per year. We also offer a five-euro discount on the EATCS membership fee to those who register both to the EATCS and to one of its chapters. Additional € 35 fee is required for ensuring the *air mail* delivery of the EATCS Bulletin outside Europe.

HOW TO JOIN EATCS

You are strongly encouraged to join (or prolong your membership) directly from the EATCS website `www.eatcs.org`, where you will find an online registration form and the possibility of secure online payment. Alternatively, contact the Secretary Office of EATCS:

Mrs. Efi Chita,
Computer Technology Institute & Press (CTI)
1 N. Kazantzaki Str., University of Patras campus,
26504, Rio, Greece
Email: `secretary@eatcs.org`,
Tel: +30 2610 960333, Fax: +30 2610 960490

If you are an EATCS member and you wish to prolong your membership or renew the subscription you have to use the Renew Subscription form. The dues can be paid via paypal and all major credit cards are accepted.

For adittional information please contact the Secretary of EATCS:

Dmitry Chistikov
Computer Science
University of Warwick
Coventry
CV4 7AL
United Kingdom
Email: `secretary@eatcs.org`,